ComponentOne

Scheduler for WPF

GrapeCity US

GrapeCity
201 South Highland Avenue, Suite 301
Pittsburgh, PA 15206
Tel: 1.800.858.2739 | 412.681.4343
Fax: 412.681.4384
Website: https://www.grapecity.com/en/
E-mail: us.sales@grapecity.com

Trademarks

Warranty

ComponentOne warrants that the media on which the software is delivered is free from defects in material and workmanship, assuming normal use, for a period of 90 days from the date of purchase. If a defect occurs during this time, you may return the defective media to ComponentOne, along with a dated proof of purchase, and ComponentOne will replace it at no charge. After 90 days, you can obtain a replacement for the defective media by sending it and a check for $2 5 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original media on which the software is delivered is set forth here, ComponentOne makes no other warranties, express or implied. Every attempt has been made to ensure that the information contained in this manual is correct as of the time it was written. ComponentOne is not responsible for any errors or omissions. ComponentOne's liability is limited to the amount you paid for the product. ComponentOne is not liable for any special, consequential, or other damages for any reason.

Copying and Distribution

Table of Contents

Scheduler for WPF and Silverlight Overview

Easily integrate Outlook-style scheduling functionality into your Silverlight and WPF apps with **Scheduler for Silverlight and WPF**. With options like customizable dialog boxes, built-in data views, import/export capabilities, and more, designing a scheduling app has never been easier.

Help with WPF and Silverlight Edition

Getting Started

- For information on installing **ComponentOne Studio WPF Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with WPF Edition.
- For information on installing **ComponentOne Studio Silverlight Edition**, licensing, technical support, namespaces and creating a project with the control, please visit Getting Started with Silverlight Edition.

Scheduler Key Features

Some of the main features of **Scheduler for WPF and Silverlight** that you may find useful include the following:

- **Select from Multiple Built-in Data Views**
  The C1Scheduler control includes five built-in data views, allowing you to offer a variety of ways for users to view their schedules. View the schedule in day, week, work week, month or time line. See Using Data Views for more information on changing the view.
- **Create Custom Views**
  You can create and use custom schedule views.
- **Bind your Schedule to the Data Source of Your Choice**
  C1Scheduler gives you the option of using standard ADO.NET data binding, the C1.**Silverlight**.Data assembly, or using a built-in datasource. Using the **Data Source settings**, which work with the C1ScheduleStorage component, you can attach a datasource and map to each column in the table to save and load appointments, categories, contacts, calendar owners, labels, resources, and the status of appointments.
- **Customize the Default Dialog Boxes**
  C1Scheduler provides options for creating your own look for Appointment, Recurrence, Reminders, and other dialog boxes. Create a copy of the default templates provided with **Scheduler for WPF and Silverlight** and customize the desired elements. You can even create an empty template and design the Appointment or other dialog boxes from scratch. See C1Scheduler Default Styles and Templates for more information on **Scheduler for WPF and Silverlight** templates.
- **Create Outlook-style Appointments**
  Users can easily add new and edit existing appointments within a C1Scheduler control. Just as in Microsoft Outlook, appointments can occur one-time or recur over a set amount of time, and reminders can be set so no appointment is missed. Additionally, C1Scheduler provides twelve built-in labels and four availability options to help users manage each appointment, as well as the ability to create custom labels. Appointments can be organized within categories, and resources and contacts for each appointment can be specified.
- **Import and Export Data**
  If you want to use a built-in datasource, you can save or load data in any supported format (binary, XML or iCal) when it is appropriate for your application. You can do it using the **C1ScheduleStorage** C1ScheduleStorage.Export and C1ScheduleStorage.Import methods or using C1Scheduler routed commands, C1ScheduleStorage.Export and C1Scheduler.Import commands.
- **Office 2007 -style Themes**
  Scheduler for WPF and Silverlight includes seven predefined themes for C1Scheduler and the calendar controls, as well as the capability to create customized themes. For more information on the built-in themes, see C1Scheduler WPF Themes, C1Scheduler Silverlight Themes, or C1Calendar Themes.

WPF Quick Start

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 C1Scheduler to your application, bind to a data source, and customize the schedule.

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

Step 1 of 4: Configuring the Data Source

In this step you will begin by creating a new project in Visual Studio 2010 and configuring the data source.

To configure the data source in Visual Studio 2010:

1. Create a new WPF project in Microsoft Visual Studio 2010:
    a. Select **File | New Project**. The **New Project** dialog box opens.
    b. Expand the **Visual C#** node and select **NET Framework 3.5**. Note that you may have to expand the **Other Languages** node to find **Visual C#**.
    c. Choose **WPF Application** from the list of *Templates* in the right pane.
    d. Enter a name for your application in the **Name** field and click **OK**. A new project is created.
2. From the **Project** menu, select **Add Page**. The **Add New Item** window appears. **Visual C#** will be selected under *Categories*, and **Page (WPF)** will be selected in the *Templates* pane.
3. Leave Page1.xaml in the **Name** text box and click **Add**. The new Page1.xaml opens.
4. If it is not already open, double-click **Page1.xaml.cs** under the Page1.xaml node in the Solution Explorer to open it.
5. Add a reference to C1.WPF.Schedule:
    a. Select **Project | Add Reference**.
    b. Browse to find the location of the **C1.WPF.dll** and **C1.WPF.Schedule** assembly files. When **Scheduler for Silverlight and WPF** is installed, this file is installed to C**:\Program Files\ComponentOne\WPF Edition\bin** by default. Note that the location may be different if you performed a custom install.
    c. Add the following using statements to the **Page1.xaml.cs** page:

    C#

    ```
    using C1.C1Schedule;
    using MYProjectNAME.ScheduleDataSetTableAdapters;
    ```

    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.
6. Add the data source you are binding to:
    a. Select **Data | Add New Data Source**. The **Data Source Configuration Wizard** appears.
    b. Select **Database** and click **Next**.
    c. Click the **New Connection** button and select **Microsoft Access DataBase File** from the Data Source group box and click **Continue**.
    d. Click **Browse** to browse to the **Schedule.mdb** database installed with **Scheduler for Silverlight and WPF**. This file was placed in the **Common** folder during installation.
    e. Click **OK** in the **Add Connection** dialog box and then click **Next** in the **Data Source Configuration Wizard** dialog box. If you are asked to copy the file to your project, you can click No. The connection is

given a name.
f. Click **Next** once you create the new connection.
g. Click **Next** to save the connection string.
h. Select the **Tables** node, leave **ScheduleDataSet** as the *DataSet name*, and click **Finish**.

7. Add the following C# code to your **Page1.xaml.cs Page1** class so it looks like the following. This code will use the data from the dataset to fill the schedule.

C#

```
public partial class Page1 : System.Windows.Controls.Page
     {
         private ScheduleDataSet _schedulerDataSet = null;
        public Page1()
         {
             InitializeComponent();
             // Use the data set to fill in the data.
             FillData(SchedulerDataSet);
         }

        public ScheduleDataSet SchedulerDataSet
         {
             get
             {
                 if (_schedulerDataSet == null)
                 {
                     _schedulerDataSet = Resources["dataSet"] as
ScheduleDataSet;
                 }
                 return _schedulerDataSet;
             }
         }

        private void FillData(ScheduleDataSet ds)
         {
             if (ds == null)
                 return;
             AppointmentsTableAdapter appAd = new AppointmentsTableAdapter();
             appAd.Fill(ds.Appointments);
         }

    }
```

8. Save and close the project.

Step 2 of 4: Binding C1Scheduler to a Data Source

Now you can add a C1Scheduler control to the Blend project and bind to the data source.

To add the new dataset as a resource to your Blend project:

1. Open Microsoft Expression Blend 4 and select **File|Open Project** from the menu to open the project (.sln) that you created in Visual Studio 2010.
2. Double-click the **Page1.xaml** in the **Project** panel to open the page and click the **Design** tab to access the **Design** view, if necessary.
3. Add a C1Scheduler control to the page. Since you have already added a reference to the C1.WPF.C1Schedule assembly

when you created the project in Visual Studio 2010, all you need to do is add C1Scheduler from the Asset Library:
   a. Click the **Asset Library** button in the Blend toolbox.
   b. In the **Asset Library**, click the **Categories** tab.
   c. Select **C1Scheduler**. The component will appear in the toolbox above the **Asset Library** button.
   d. Double-click the C1Scheduler button in the toolbox to add it to the page.
4. Click the **XAML** tab to switch to **XAML** view.
5. Create a CLR namespace by adding the following XAML code to the namespace list within the **Page** tag. Use your Visual Studio 2010 project's name for the clr-namespace value.
   `xmlns:local="clr-namespace:MYProjectNAME"`

   Your XAML should look similar to the following:

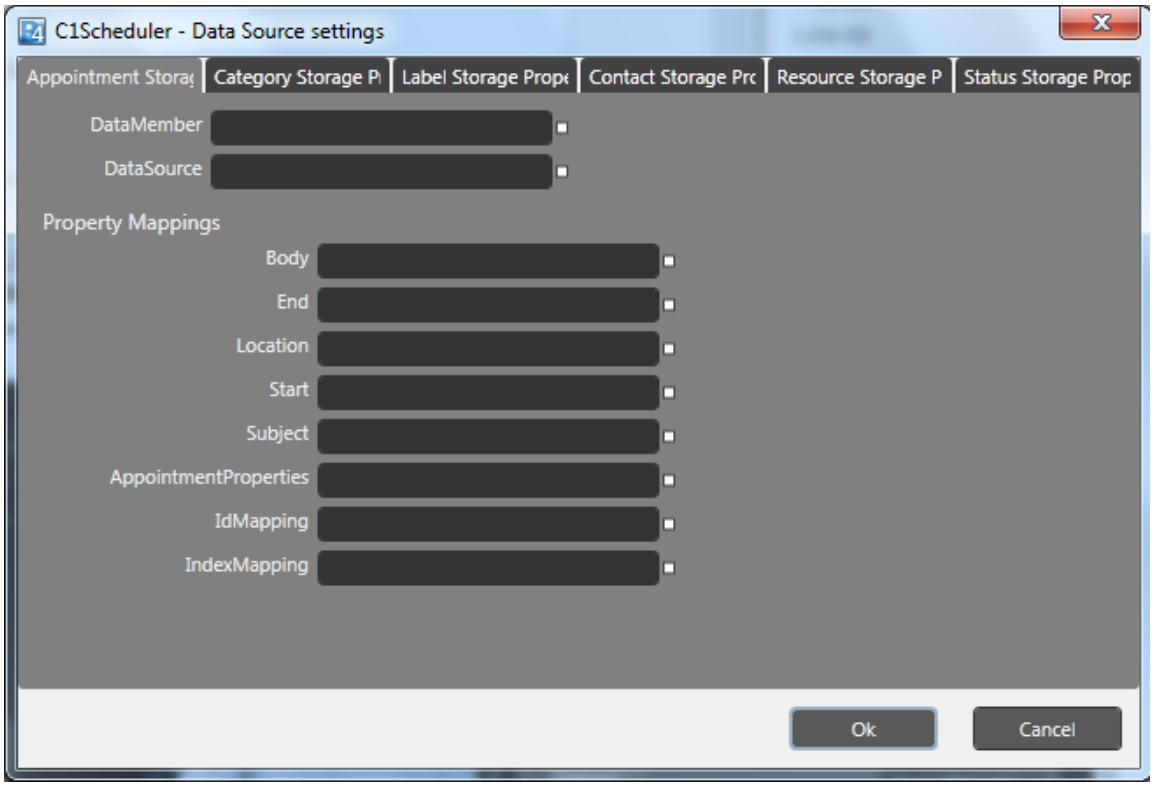
```
XAML
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="MYProjectName.Page1"
    Title="Page1"
    xmlns:c1sched="http://schemas.componentone.com/wpf/Schedule"
x:Class=”WpfScheduleQuickStart.Page1”
   xmlns:local="clr-namespace:MYProjectName">
```

6. Right-click Page1.xaml in the **Project** panel and select **Startup**. This will set Page1 as the opening window.
7. Select **Project | Build Project** to build your project.
8. While you are still in XAML view, add the ScheduleDataSet as a resource by entering the following XAML immediately after the XAML above:

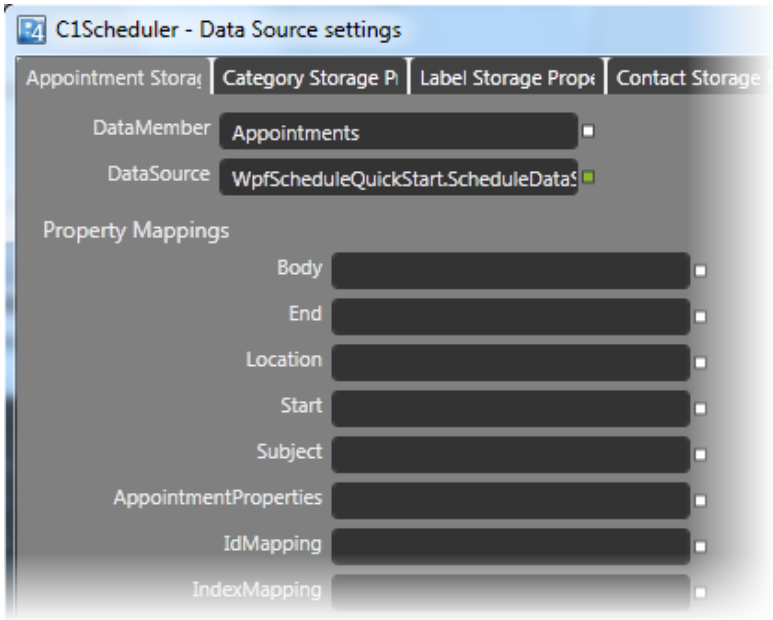
```
XAML
<Page.Resources>
    <local:ScheduleDataSet x:Key="dataSet" />
</Page.Resources>
```

To map to the Scheduler for WPF Data Storage:

1. In your Blend project, click the **Design** tab to switch back to **Design** view of Page1.xaml.
2. Select the C1Scheduler control on **Page1.xaml** of your project.
3. In the **Properties** panel, under **Data**, click the **DataStorage** ellipsis button. The **Data Source settings** dialog box appears.



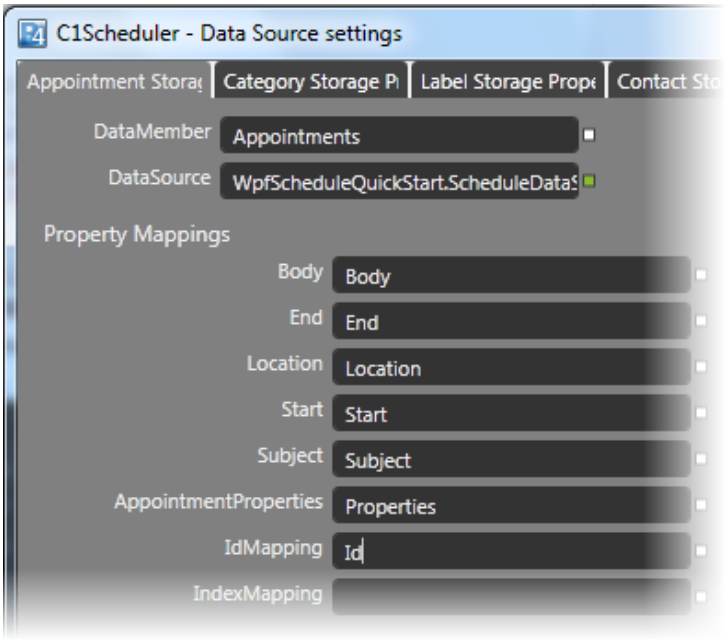
4. Click the **Advanced Property Options** button next to the **DataSource** property, select **Local Resource** and check **dataSet**. This property is set to the **ScheduleDataSet**.
5. Enter **Appointments** next to the **DataMember** property. C1Scheduler will map to the **Appointments** table and use its data to fill in the schedule.



6. Next, set the **mappings** for the properties to the corresponding data fields in the **Appointments** table. Enter the following text for each of the **Property Mappings** items:

Property	**Text**
Body	Body

End	End
Location	Location
Start	Start
Subject	Subject
AppointmentProperties	Properties
IdMapping	Id
IndexMapping	



7. Click **OK** to close the **Data Source settings** dialog box. The database is now mapped to the **Appointment Storage**. The XAML code for the mappings looks similar to the following:

XAML

```
<c1:C1Scheduler HorizontalAlignment="Left" Height="400" VerticalAlignment="Top"
Width="480">
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataMember" Value="Appointments"/>
            <c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataSource" Value="{DynamicResource
dataSet}"/>
            <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:C1Scheduler>
```

You have successfully bound C1Scheduler to a data source. Now you can customize the schedule.

Step 3 of 4: Selecting a Data View and Theme

In this step you will select one of the predefined data views and themes.

To select a Data View:

1. Make sure C1Scheduler is still selected.
2. In the **Properties** panel, under **View**, click the drop-down arrow next to the BeforeViewChangeEventargs.Style property and select **Reset**.
3. Click the drop-down arrow again and select **Week View**.

To select a Theme:

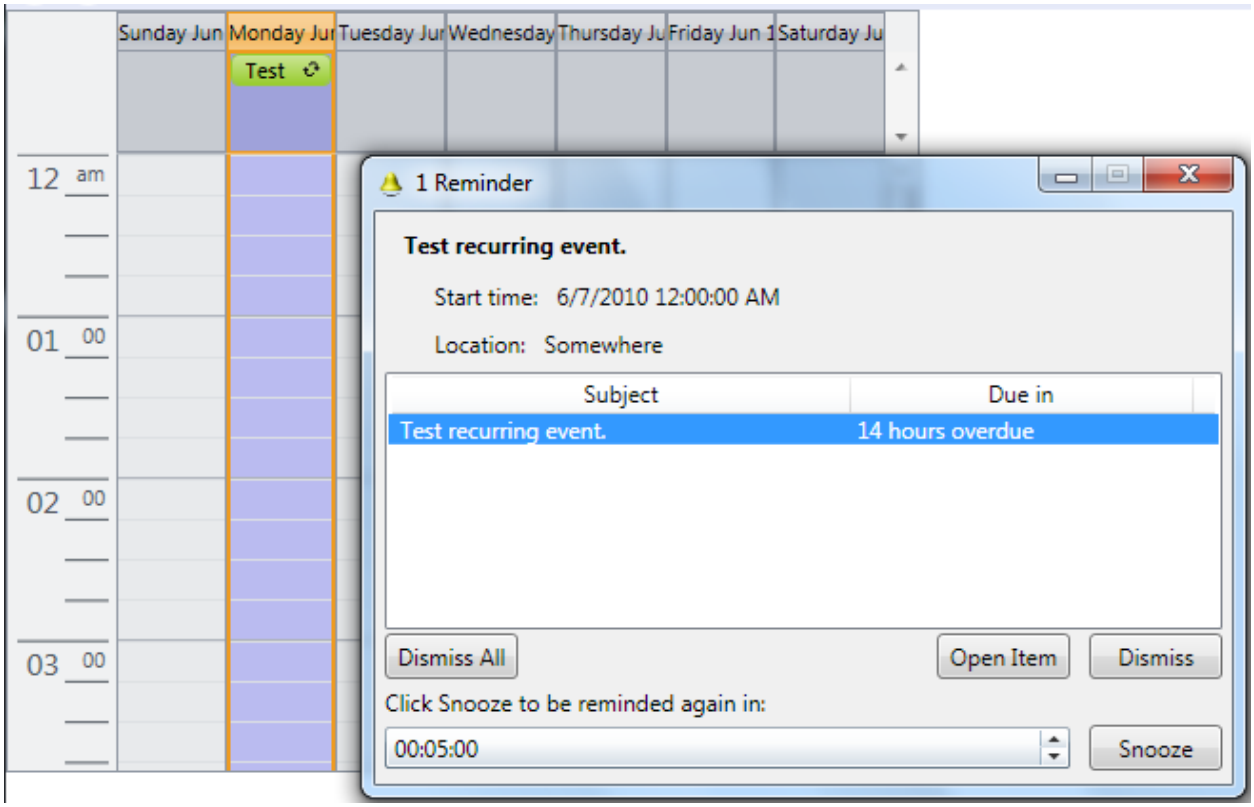
Expand the **View** node, click the drop-down arrow next to C1Scheduler.Theme, and select **Office 2007 Black**.

Step 4 of 4: Running the Application

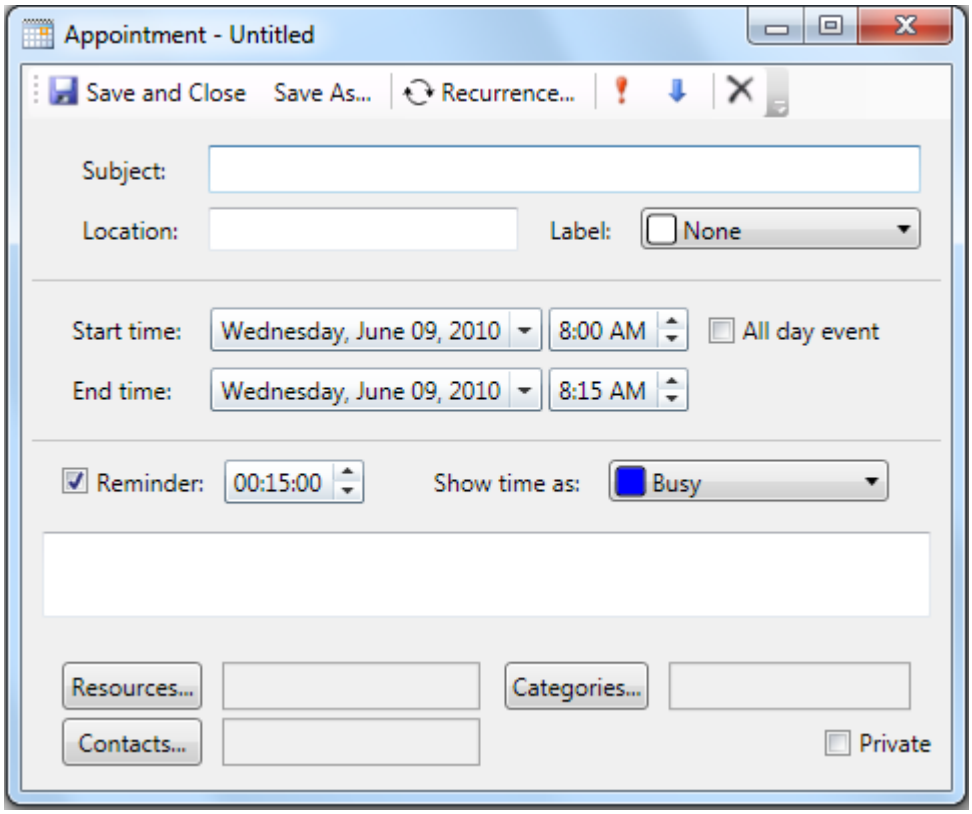
Now that you've created a schedule application, bound the schedule to a datasource, and customized the schedule's appearance in Blend, the only thing left to do it run the application.

To run the schedule application and observe Scheduler for WPF's run-time behavior:

1. Press **F5** or select **Test Solution** from the **Project** menu. The schedule and a **Reminder** dialog box appears. Click **Dismiss All**.

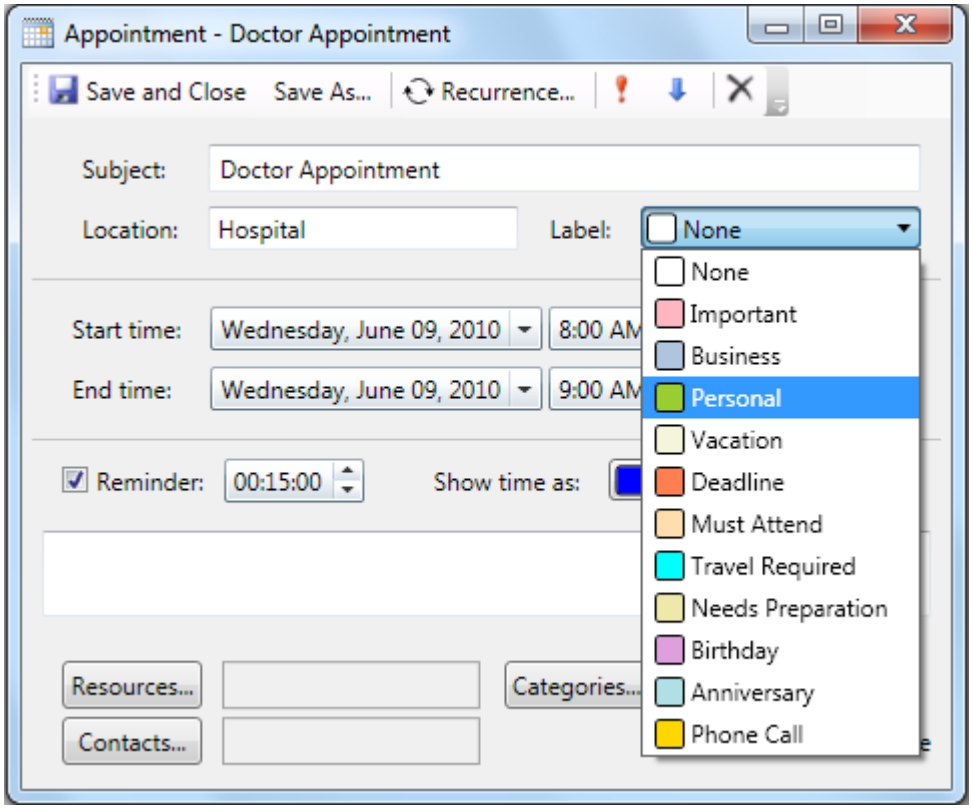


2. Set up an appointment by double-clicking the time for the appointment under the desired day and time. In this example, create an appointment for 8AM on Wednesday, June 9th. The **Appointment** dialog box opens.

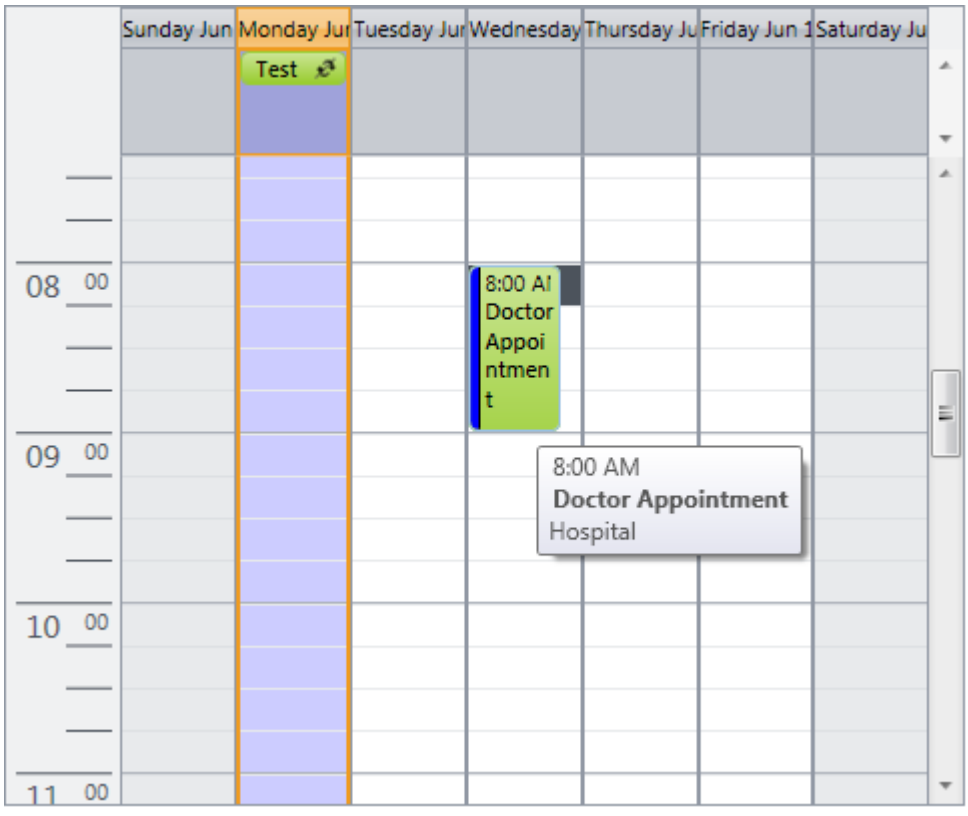


3. Enter “Doctor Appointment” for the **Subject**, “Hospital” for the **Location**, and “9:00AM” for the **End time**.

4. Click the drop-down arrow next to the **Label** and select **Personal**.



5. Click the **Save and Close** button. The appointment now appears in the schedule.



Congratulations! You've completed the quick start and created a **WPF** application, bound the schedule to a data source, and created a new appointment.

Silverlight Quick Start

The following quick start guide is intended to get you up and running with **Scheduler for Silverlight and WPF**. In this quick start you'll create a new Silverlight project, add C1Scheduler to your application, set the data view, and add navigation buttons to the schedule.

Step 1 of 4: Creating a Silverlight application

Begin by creating a Silverlight application.

1. Create a new Silverlight project in Microsoft Visual Studio:
    a. Select **File | New | Project**. The **New Project** dialog box opens.
    b. Expand the **Visual C#** node and select **NET Framework 4.0**. Note that you may have to expand the **Other Languages** node to find **Visual C#**.
    c. Select **Silverlight** from the list of Installed Templates in the left pane.
    d. Choose **Silverlight Application** from the list of *Templates* in the right pane.
    e. Enter a name for your application, *SchedulerQuickStart* for example, in the **Name** field and click **OK**.
    f. Click **OK** again. A new project is created.

Step 2 of 4: Setting the Data View

In this step you will select use one of the predefined data views.

1. In the Visual Studio **Project** menu, select **Add Reference**.
2. Browse to find the **C1.Silverlight.dll** and **C1.Silverlight.Schedule.dll**. These .dlls are installed by default in the **C:\Program Files\ComponentOne\Silverlight Edition** folder; however, they may appear elsewhere depending on where you installed the product. Click **OK** once you've selected these .dlls.
3. Double-click the MainPage.xaml file in the Visual Studio Solution Explorer.
4. Add the **Scheduler** control from the toolbox to the MainPage.xaml.
5. Add XAML markup so the MainPage.xaml XAML looks like the following:

XAML

```
<UserControl
    xmlns:c1="clr-
namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule"
    x:Class="SchedulerQuickStart.MainPage"
    xmlns=http://schemas.microsoft.com/winfx/2006/xaml/presentation
    xmlns:x=http://schemas.microsoft.com/winfx/2006/xaml
    xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc=http://schemas.openxmlformats.org/markup-compatibility/2006
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">

    <Grid x:Name="LayoutRoot">
        <c1:C1Scheduler x:Name="sched1" Style="{Binding OneDayStyle,
RelativeSource={RelativeSource Self}}"/>
    </Grid>
</UserControl>
```

When you run the project, it will show a schedule in Day View. In the next step, you will add navigation buttons.

Step 3 of 4: Adding Navigation Buttons

In this step you will add navigation buttons to switch between the C1Scheduler views. Add the following markup between the <Grid> tags and above the <c1sched> tags you added in the previous step:

XAML

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

<StackPanel Grid.Row="0" Orientation="Horizontal">

<Button Content="One Day"
    c1:CommandExtensions.CommandTarget="{Binding ElementName=sched1}"
    c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
    c1:CommandExtensions.CommandParameter="{Binding Path=OneDayStyle,
ElementName=sched1}"/>
<Button Content="Working Week"
    c1:CommandExtensions.CommandTarget="{Binding ElementName=sched1}"
    c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
    c1:CommandExtensions.CommandParameter="{Binding Path=WorkingWeekStyle,
ElementName=sched1}"/>

<Button Content="Week"
    c1:CommandExtensions.CommandTarget="{Binding ElementName=sched1}"
    c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
    c1:CommandExtensions.CommandParameter="{Binding Path=WeekStyle,
ElementName=sched1}"/>

<Button Content="Month"
    c1:CommandExtensions.CommandTarget="{Binding ElementName=sched1}"
    c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
    c1:CommandExtensions.CommandParameter="{Binding Path=MonthStyle,
ElementName=sched1}"/>
<Button Content="Time Line"
    c1:CommandExtensions.CommandTarget="{Binding ElementName=sched1}"
    c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
    c1:CommandExtensions.CommandParameter="{Binding Path=TimeLineStyle,
ElementName=sched1}"/>
</StackPanel>
```

The attached properties of the **C1.Silverlight.CommandExtensions** class will handle button clicks and execute corresponding C1Scheduler commands.

Step 4 of 4: Running the Application

To run the application and view the schedule:

1. Press **F5**. The schedule appears.
2. Click the **One Day**, **Working Week**, **Week**, **Month or TimeLine** button to change the schedule view.

Congratulations! You've completed the **Scheduler for Silverlight** quick start and created a **Scheduler for Silverlight** application.

XAML Quick Reference

This section provides a few examples that show how to use the **C1Scheduler** control with only XAML code.

EX: Assign Values to a Nested Property

XAML does not provide the ability to assign a value to a nested property. However, the object model of the C1Scheduler.DataStorage contains nested properties by nature. To work around this limitation, the NestedPropertySetter class can be used.

Note: The NestedPropertySetter class works in a conjunction with C1Scheduler only.

Elements of this class, being placed as children of a C1Scheduler element in XAML, represent setters as **Property**/**Value** pairs where the **Property** specifies a property path relative to a parent C1Scheduler element. In the following example, the C1BindingSource.DataMember property is assigned to the **Appointments** table of a database. The C1BindingSource.DataSource is set to the data set resource in the project:

XAML

```
<c1:C1Scheduler >
        <!-- Map AppointmentStorage -->
        <c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataMember" Value="Appointments"/>
        <c1:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataSource" Value="{StaticResource
dataSet}"/>
</c1:C1Scheduler >
```

EX: Set the Days of the Week for Working Week View

The following XAML sets the days of the week for Working Week View:

XAML

```
 <c1:C1Scheduler x:Name="scheduler1" Theme="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=Office2007.Default}}"
Style="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=WorkingWeekStyle}}">
       <c1:C1Scheduler.CalendarHelper>
             <c1:CalendarHelper Culture="English " WeekStart="Sunday"
                    EndDayTime="18:20:00" StartDayTime="09:20:00"
                    WorkDays="Tuesday,Wednesday,Thursday,Friday,Saturday">
             </c1:CalendarHelper>
       </c1:C1Scheduler.CalendarHelper>
  </c1:C1Scheduler>
```

EX: Bind C1Scheduler to C1Calendar

The following XAML code binds C1Scheduler to a C1Calendar control:

XAML

```
<c1:C1Calendar HorizontalAlignment="Left" Margin="34,51,0,0" Name="c1Calendar1"
VerticalAlignment="Top" SelectedDate="{Binding Path=SelectedDateTime,
ElementName=C1Scheduler1, Mode=TwoWay}" />
```

Scheduler Components and Controls

Scheduler for Silverlight and WPF consists of the following controls and components, which provide all of the functionality for a scheduling application:

The C1Scheduler Control

The C1Scheduler control is a fully functional schedule that allows users to add, edit, and manage their appointments. It is composed of two main parts:

- A storage area that contains business data, such as appointments, contacts, and so on, with an ability to bind this data to database storages like DataSet, and provides business logic for this data. The storage is referenced by the C1Scheduler.DataStorage property and is represented by the C1ScheduleStorage root class. The object model of C1ScheduleStorage is the same for all ComponentOne scheduling tools for different platforms, such as WinForms and ASP.NET.
- The default C1Scheduler interface, which is intended to help in building an arbitrary user interface for data managed by C1Scheduler.

The C1Calendar Control

C1Calendar control is a part of C1.WPF.Schedule assembly.

This is a refactored version of the old **C1MultiMonthCalendar** control. It is used to create a calendar in XAML, showing multiple months and allowing users to select a specific date interactively. Some properties such as Time and DateTime have been removed from C1Calendar.The Date property has been renamed to the SelectedDate, etc.

The C1CalendarItem control can be used only as a part of C1Calendar control and won't work as a standalone control.

The C1ScheduleStorage Component

The C1ScheduleStorage component handles all of the data operations for the C1Scheduler control. Users do not have to create the **C1ScheduleStorage** component from the code, as it is automatically created by the **C1Scheduler** control.

The **C1ScheduleStorage** component contains specific data storages for appointments, resources, contacts, labels, and statuses, which work on the data layer to provide data to the scheduling application. The **C1ScheduleStorage** component contains some separate data storages as listed in the following table:

Data Storage	Description
AppointmentStorage	Storage for appointment data.
CategoryStorage	Storage for category data.
ContactStorage	Storage for contact data.
OwnerStorage	Storage for calendar owners data.
LabelStorage	Storage for label data.
StatusStorage	Storage for status data.
ResourceStorage	Storage for resource data.

When providing data to these storages, it is necessary to specify mappings between data fields in the datasource and their interpretation in the storage.

All storages are inherited from the BaseStorage class, which provides all the basic functionality required for data binding. To bind a storage to a specific data table (or another data object) it is necessary to set its C1BindingSource.DataSource and C1BindingSource.DataMember properties, and then set the mappings for the properties to corresponding data fields in the bound datasource. For example, if the bound data source contains a field called *EmployeeName*, you should set the ContactStorage.Mappings.TextMapping.MappingInfo.MappingName property to the name of this field.

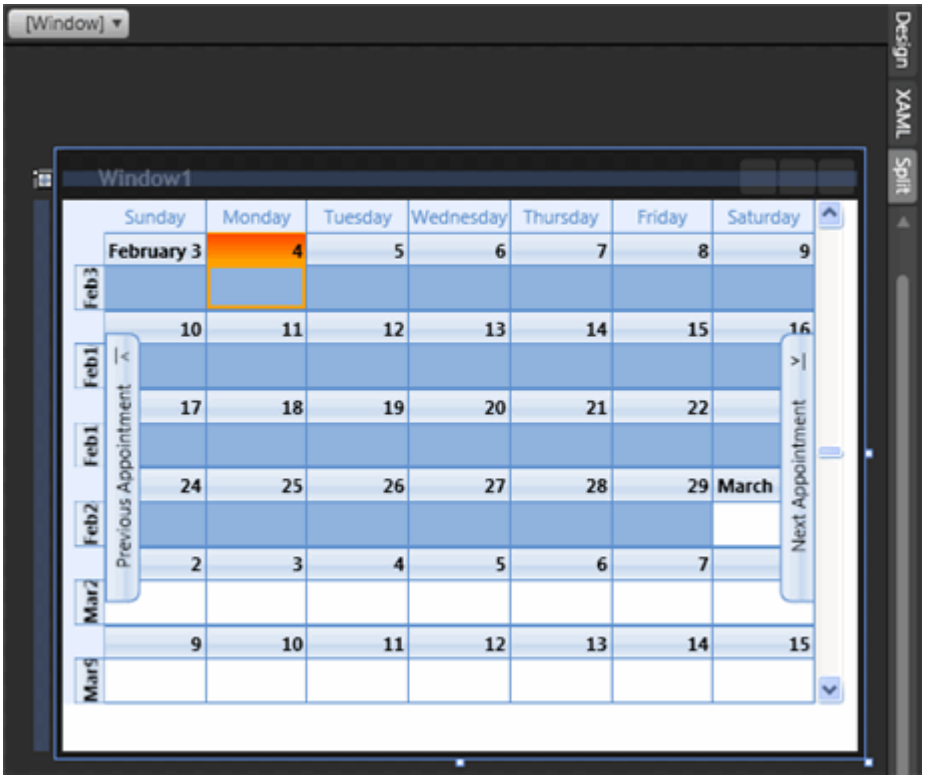
The **BaseStorage** component has almost the same functionality as the **System.Windows.Forms.BindingSource** component, but does not depend on **System.Windows.Forms** namespace.

Note: The CategoryStorage, LabelStorage and StatusStorage storages contain a predefined set of standard items.

Using the C1Scheduler Control

When you place a C1Scheduler control into a XAML window, it is a functional schedule that allows users to add, edit

and manage appointments. However, the initial schedule uses a default interface that you will want to further customize to fit your users' scheduling needs. The default user interface looks like the following image in Microsoft Blend **Design** view:



Scheduler for Silverlight Appearance

There are several ways you can customize **C1Scheduler**'s appearance. The following topics discuss some of these customization techniques, including data views, templates, and themes.

Using Data Views

C1Scheduler provides a number of views, such as **Day** view or **Month** view, to display data in the C1Scheduler.DataStorage.

Data View	Description
One Day View	Displays a detailed view showing appointments for a particular day.
Month View	Displays appointments for one or more months. This is the default view.
Week View	Displays appointments for specified week days.
Working Week View	Displays appointments for any given weekly period. The default is Monday through Friday.
Time Line View	Displays a time line.

Each view is represented by a single style, each of which is fully defined in XAML. These predefined styles can be applied to C1Scheduler and are accessible via **C1Scheduler properties**: *OneDayStyle*, *WorkingWeekStyle*, *WeekStyle*, *MonthStyle* and *TimeLineStyle*. The default view of the interface is *MonthStyle*, and the current date is shown.

To specify a view, **Day View** for example, assign scheduler's C1Scheduler.ChangeStyle method using the C1Scheduler.OneDayStyle property value:

Visual Basic

```
Scheduler1.ChangeStyle(Scheduler1.OneDayStyle)
```

C#

```
Scheduler1.ChangeStyle(Scheduler1.OneDayStyle);
```

XAML

```
<c1sched:C1Scheduler Style="{Binding OneDayStyle, RelativeSource={RelativeSource Self}}"/>
```

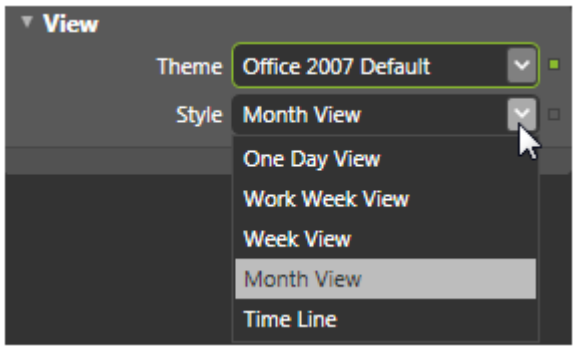
The data view can be set at design time in Visual Studio using the context menu:

1. Right-click the C1Scheduler control.
2. Select **View** and choose **One Day**, **Work Week**, **Week**, or **Month**.

Note: You can also change the view through the Properties window by selecting the option from the drop-down list next to the **Style** property.

The style can also be set in the Design view of Microsoft Blend. To change the visual Style in a Microsoft Blend project:

1. Select the C1Scheduler control in your XAML window or page.
2. In the Properties panel, under View, click drop-down arrow next to the **C1Scheduler.Style** property.
3. Select one of the available view styles.



These properties are used by C1Scheduler to automatically change the styles used if the range of visible dates is changed by the associated calendar control or if the range of visible dates and/or view type is changed by a C1Scheduler command. The user can alter the default behavior by handling the C1Scheduler.BeforeViewChange event or by setting the **C1Scheduler** ViewStyleSelector property to the custom **StyleSelector** object.

To change the whole style set used by **C1Scheduler**, set the style properties to custom defined styles. For example:

C#

```
// set C1Scheduler's styles to a custom look.
scheduler1.OneDayStyle = customOneDayStyle;
scheduler1.WorkingWeekStyle = customWorkingWeekStyle;
```

```
scheduler1.WeekStyle = customWeekStyle;
scheduler1.MonthStyle = customMonthStyle;
```

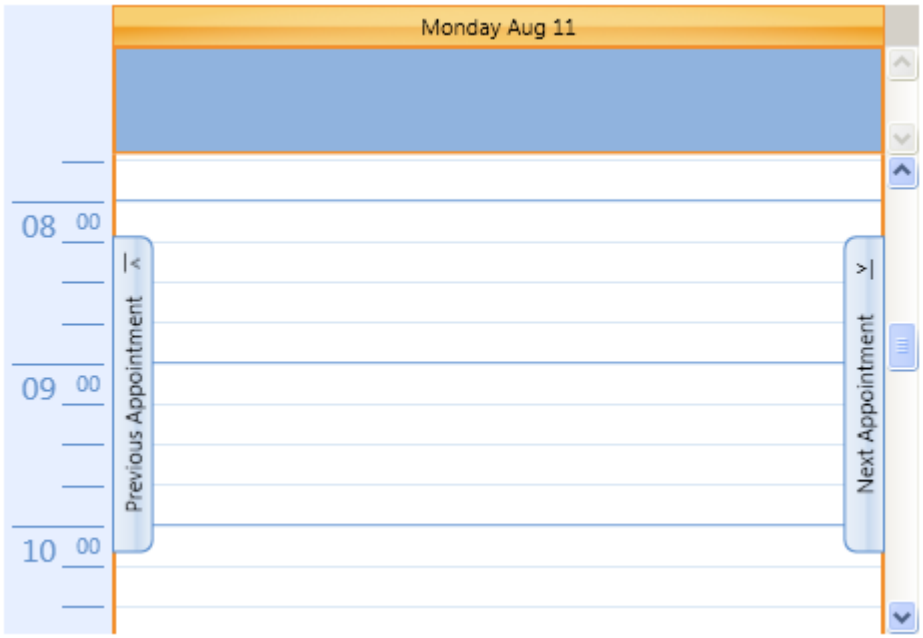
Data View Keys and Appearance

Default view styles are defined with the following resource keys:

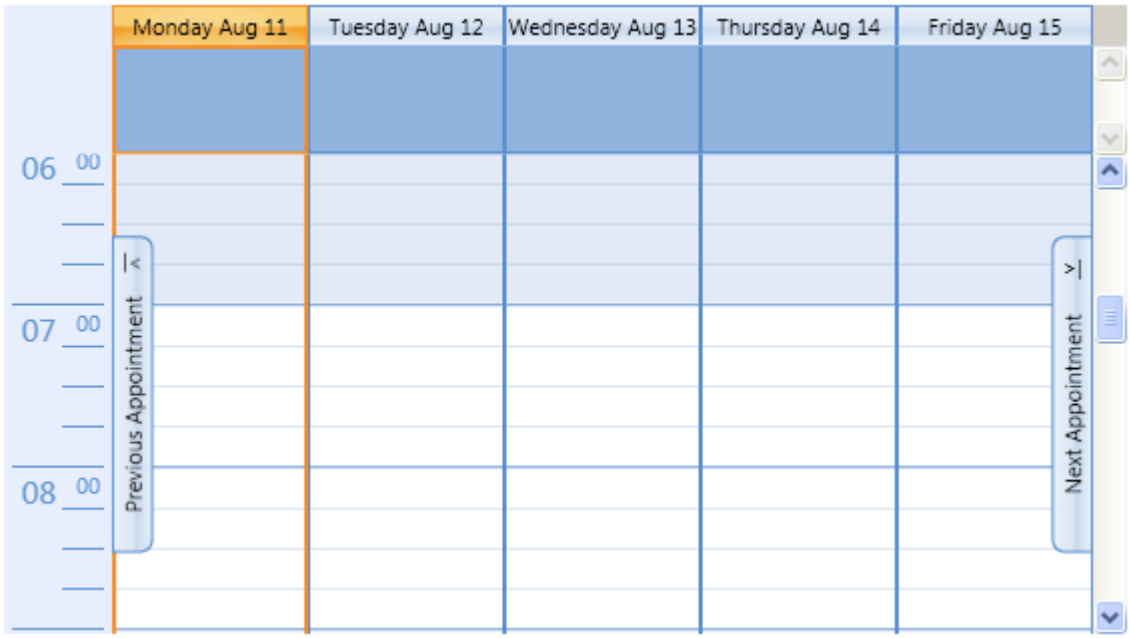
WPF ComponentResourceKey	Silverlight Resource Key	Description
x:Key="{ComponentResourceKey TypeInTargetAssembly={Type local:C1Scheduler}, ResourceId=OneDayStyle}" TargetType="{x:Type local:C1Scheduler}"	x:Key="OneDayStyle" TargetType="c1sched:C1Scheduler"	One Day View
x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=WorkingWeekStyle}" TargetType="{x:Type local:C1Scheduler}"	x:Key="WorkingWeekStyle" TargetType="c1sched:C1Scheduler"	Work Week View
x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=WeekStyle}" TargetType="{x:Type local:C1Scheduler}"	x:Key="WeekStyle" TargetType="c1sched:C1Scheduler"	Week View
x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=MonthStyle}" TargetType="{x:Type local:C1Scheduler}"	x:Key="MonthStyle" TargetType="c1sched:C1Scheduler"	Month View
x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=TimeLineStyle}" TargetType="{x:Type local:C1Scheduler}"	x:Key="TimeLineStyle TargetType="c1sched:C1Scheduler"	Time Line View

The following images show examples of the appearance for each of the available data views.

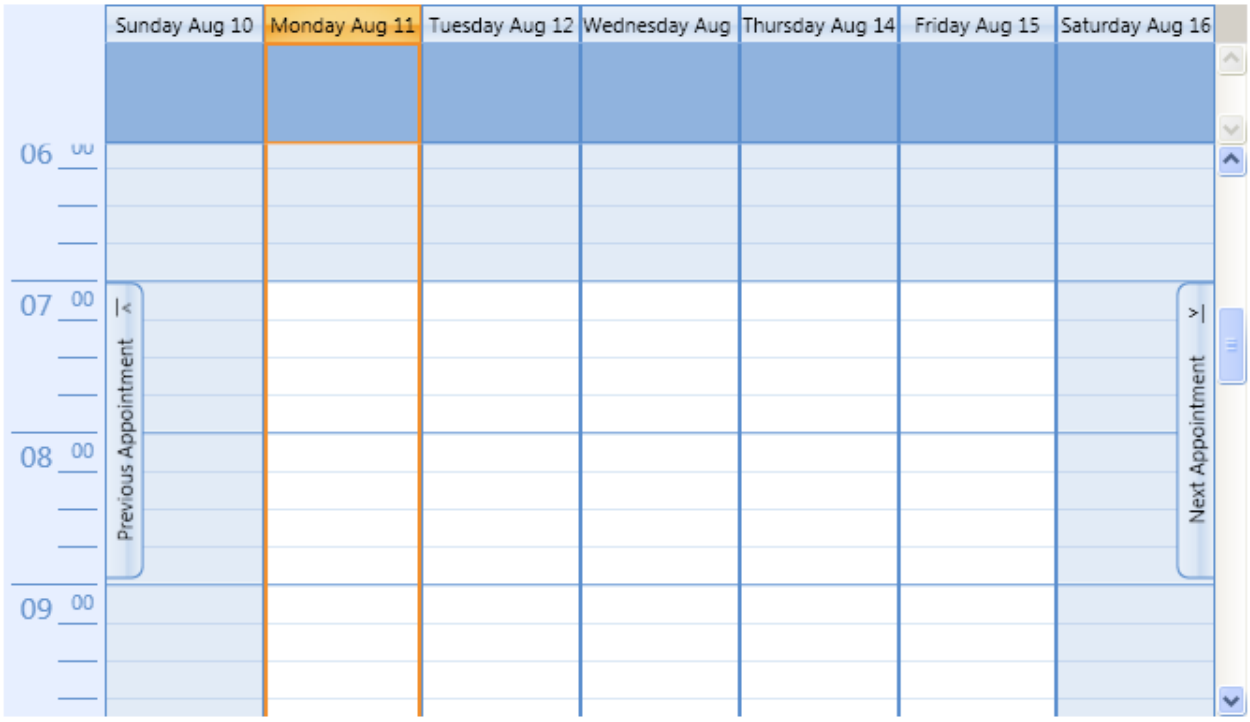
One Day View



Working Week View

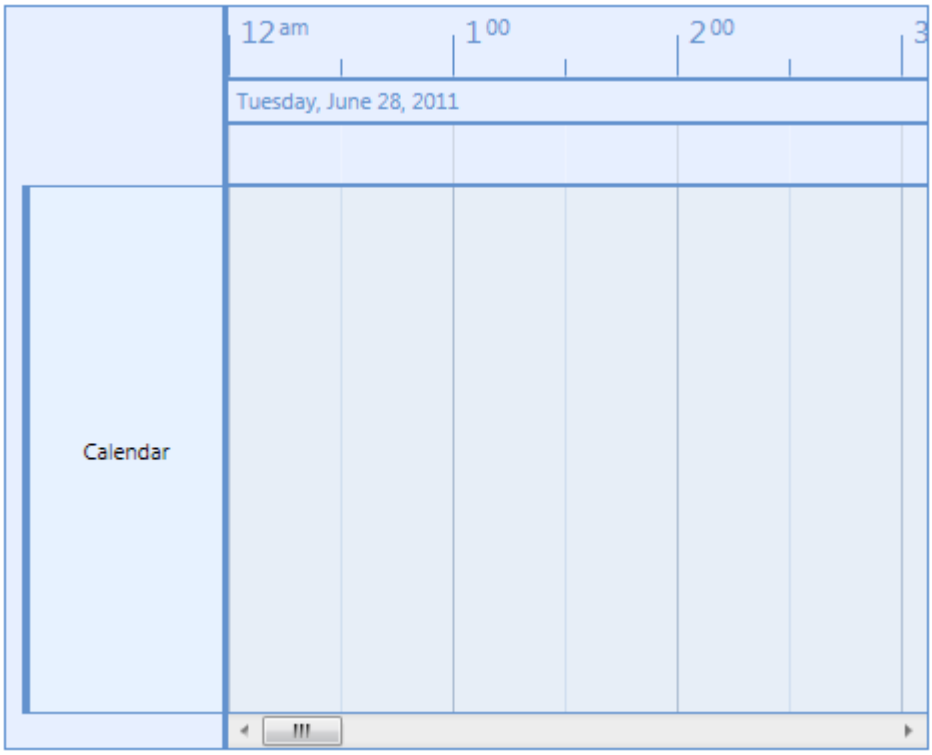


Week View



Month View

Time Line View



C1Scheduler Default Styles and Templates

Default styles are defined using styles and templates for smaller C1Scheduler parts. The following table depicts the details of default C1Scheduler styles and templates along with their keys:

Keys used by all default themes

WPF

Resource key	Description
PART_C1SchedulerScrollBar	Determines the style of the scroll bar.
PART_C1Scheduler_WorkHour_Style	Determines the style of the one work hour group in a day view.
PART_C1Scheduler_FreeHour_Style	Determines the style of the one free hour group in a day view.
PART_C1Scheduler_WorkSlot_Template	Determines the template of a single free time slot in a day view.
PART_C1Scheduler_FreeSlot_Template	Determines the template of a single free time slot in a day view.
C1Scheduler_AllDayArea_Template	Determines the template used for displaying the All-Day area in a Day view.
C1Scheduler_TimeRuler_Template	Determines the template used for one hour of a time ruler in a Day view.
C1Scheduler_MonthHeader_Style	Determines the style of the month grid header (week day names).
C1Scheduler_OneMonthDay_Template	Determines the template used for displaying one day in a Month View (includes day header and day content).
C1Scheduler_OverflowJumper_Template	Determines the template used for displaying overflow jumper in a Month View and Office 2003 Week View when not all appointment elements fit

	into available day space.
PART_GroupNavigationScrollBar	If PART_GroupNavigationScrollBar is present in the control template, C1Scheduler will use it for the group navigation. This part is included into the default TimeLine style.

Silverlight

Resource Key	Description
C1Scheduler_PreviousButton_Style	Determines a style of Previous button used in previous/next navigation panel.
C1Scheduler_NextButton_Style	Determines a style of Next button used in previous/next navigation panel.
C1Scheduler_PrevNextAppPane_Style	Defines a style for previous/next appointment navigation pane (containing next/previous labels) represented by ContentControl.
IntervalAppointmentPresenterStyle	Determines a style used to display single appointment represented by IntervalAppointmentPresenter control.
IntervalAppointmentTemplate	Determines a template, used to display single appointment.
BaseViewStyle	Determines a base style for all C1Scheduler views.
C1Scheduler_TimeRuler_Template	Determines the template used for one hour of a time ruler in a Day view.
TimeSlotGroupStyleSelector	Selects the style for displaying working or free hours.
DayGroupStyleSelector	Selects the style for displaying ordinal or today days.
DayHeaderStyleSelector	Selects the style used for displaying day header in a Day view.
TimeSlotStyleSelector	Determines the style used for displaying individual time slots in a Day view.
C1Scheduler_AllDayArea_Template	Determines the template used for displaying All-Day area in a Day view.
AllDayAreaStyleSelector	Determines the style used for displaying All-Day area in a Day view.
C1Scheduler_MonthHeader_Style	Determines the style of the month grid header (week day names).
C1Scheduler_OverflowJumper_Style	Determines the style used for displaying overflow jumper in a Month View when not all appointment elements fit into available day space.
DayHeaderButtonStyle	Determines style used for displaying day header button for ordinal day in a Month view.
TodayHeaderButtonStyle	Determines style used for displaying today header button in a Month view.
MonthDayStyleSelector	Determines the style used for displaying single day in a Month view.
C1Scheduler_WeekTab_Style	Determines the style of the week tab in a Month view.
PART_GroupNavigationScrollBar	If PART_GroupNavigationScrollBar is present in the control template, C1Scheduler will use it for the group navigation. This part is included into the default TimeLine style.

Office 2007 specific keys

Resource key	Description

C1Scheduler_PrevNextAppPane_Style	Defines a style for previous/next appointment navigation pane (containing next/previous labels) represented by ContentControl.
C1Scheduler_PrevNextAppPane_Style	Determines the style of the day group in a day view.
C1Scheduler_PrevNextAppPane_Style	Determines the style of the day group in a day view.
C1Scheduler_PrevNextAppPane_Style	Determines the style of the week tab in a Month view.

Note that using these resource keys is not obligatory. You can use any keys and assign your custom styles to the corresponding C1Scheduler properties. For example:

XAML

```
<my:C1Scheduler MonthStyle="{DynamicResource customOneDayStyle}}"
WeekStyle="{DynamicResource customWeekStyle}}"/>
```

C1Scheduler's default interface includes some default **DataTemplate** objects. Custom user interfaces for this template can be provided as a **DataTemplate** object, which should be assigned to an appropriate C1Scheduler property. The default DataTemplates is accessible through ComponentResourceKeys. The following table lists the C1Scheduler property that defines its **DataTemplate** and the default **DataTemplate ComponentResourceKey**.

C1Scheduler Property	Default DataTemplate	Description
IntervalAppointmentTemplate	x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceID=IntervalAppointmentTemplate}"	Appointment element template.

Using C1Scheduler Default Templates

C1Scheduler's default interface includes a number of default **DataTemplate** objects, such as the **Appointment**, **Edit**, and **Reminder** dialog boxes. Custom user interfaces for these templates can be provided as **DataTemplate** objects, which should be assigned to an appropriate C1Scheduler property. The following table lists the C1Scheduler property that defines its **DataTemplate** and the default **DataTemplate**.

C1Scheduler Property	Default DataTemplate	Description
EditAppointmentTemplate	<DataTemplate> <c1sched:EditAppointmentControl/> </DataTemplate>	Edit/New Appointment dialog box template.
EditRecurrenceTemplate	<DataTemplate> <c1sched:EditRecurrenceControl /> </DataTemplate>	Edit Recurrence dialog box template.
SelectFromListTemplate	<DataTemplate> <c1sched:SelectFromListScene /> </DataTemplate>	Select Resources/Categories/Contacts dialog box template.
ShowRemindersTemplate	<DataTemplate> <c1sched:ShowRemindersControl /> </DataTemplate>	Reminders window template.

Using ClearStyle technology

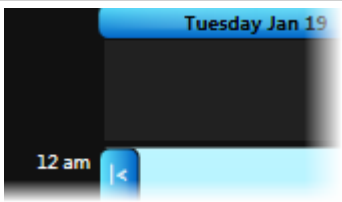
C1Scheduler supports ComponentOne's ClearStyle technology that allows you to easily change control colors without having to change control templates. With just setting a few color properties in you can quickly style the entire control appearance. The following table lists common C1Scheduler properties which can be used to customize control appearance.

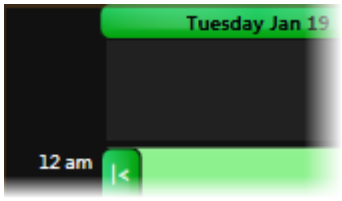
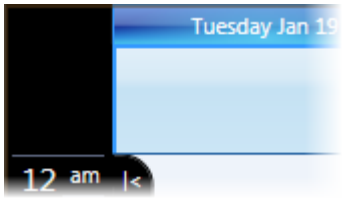
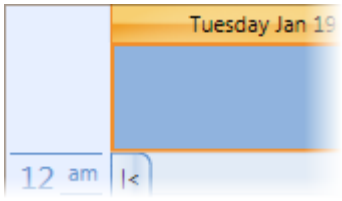
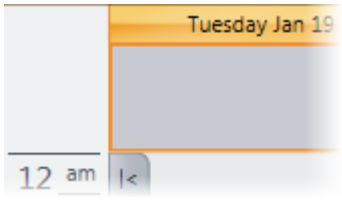
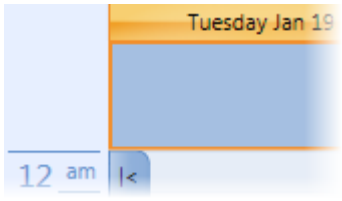
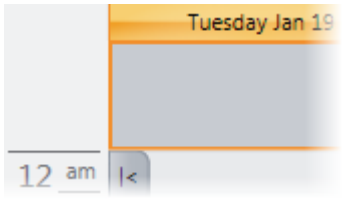
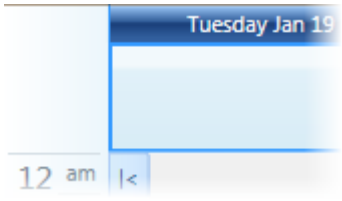
Property	Description
Background	Gets or sets a Brush that is used as a base palette color for individual days and time slots.
Foreground	Gets or sets a Brush that is the text color in day headers, week tabs and navigation panels.
ControlBackground	Gets or sets a Brush that is the face color of a control (Time Ruler, Month View header, etc.).
ControlForeground	Gets or sets a Brush that is the text color in a control (Time Ruler, Month View header, etc.).
AppointmentForeground	Gets or sets a Brush used as a foreground for appointment elements.
AlternatingBackground	Gets or sets a Brush used as background for working hours and days of alternating months.
SelectedBackground	Gets or sets a Brush used to highlight currently selected day.
TodayBackground	Gets or sets a Brush object used to color current day.

C1Scheduler WPF Themes

There are many options when it comes to setting themes for C1Scheduler. Themes can be set at design time, in XAML using the ResourceID, in code using the C1SchedulerResources static fields, or you can define a **ResourceDictionary** and **DefaultTheme** key in your Page, Window, or Application resources. The most common user interface properties of the C1Scheduler control, such as border and background brushes, default styles, and so on are defined in theme ResourceDictionaries.

C1Scheduler includes seven predefined themes.

C1Scheduler Themes	Example
Dusk Blue	

Dusk Green	
Media Player	
Office 2007 Default (Office 2007 Blue)	
Office 2007 Black	
Office 2007 Blue	
Office 2007 Silver	
Vista	

The tables below list the name of the .xaml file that contains the theme definition, the C1SchedulerResources static fields that can be used to set the theme, the ResourceIDs that can be used to set the theme, and a description of each theme.

Note: By default, the themes are installed in the folder specified below within the C:\Program Files\ComponentOne\WPF\C1WPFNewSchedule\XAML\themes\SchedulerThemes folder.

Office 2007 Themes

The following themes are in the **Office2007** folder.

Theme File	Static field of C1SchedulerResources class	ResourceID	Description
Default.xaml	Office2007Default	Office2007.Default	Office 2007 Default theme. It uses semi-transparent colors, allowing you to changes the control's palette by changing the Background property. This file contains the full definition of Office 2007 styles and templates which are used by other Office 2007 themes.
Black.xaml	Office2007Black	Office2007.Black	Office 2007 Black theme.
Blue.xaml	Office2007Blue	Office2007.Blue	Office 2007 Blue theme. This file contains full definition of Office 2007 styles and templates which are used by other Office 2007 themes.
Silver.xaml	Office2007Silver	Office2007.Silver	Office 2007 Silver themes.

Dusk Themes

The following themes are in the **Dusk** folder.

Theme File	Static field of C1SchedulerResources class	ResourceID	Description
Blue.xaml	DuskBlue	Dusk.Blue	Dusk Blue theme.
Green.xaml	DuskGreen	Dusk.Green	Dusk Green theme.

Media Player Theme

The following themes are in the **Media Player** folder.

Theme File	Static field of C1SchedulerResources class	ResourceID	Description
MediaPlayer.xaml	MediaPlayer	MediaPlayer	Media Player theme.

Vista Theme

The following themes are in the **Vista** folder.

Theme File	Static field of C1SchedulerResources class	ResourceID	Description
Vista.xaml	Vista	Vista	Vista theme.

Setting the C1Scheduler Theme

When C1Scheduler is first added to your project, it is formatted with the **Office 2007 Default** theme. If you want to use a different theme, there are several ways to select a new one.

To set the theme at design time in Visual Studio:

1. Right-click the **C1Scheduler** control.
2. Select **Theme** and choose one of the seven predefined themes.

> **Note:** You can also change the theme through the Properties window by selecting the option from the drop-down list next to the C1Scheduler.Theme property.

To set the theme in Microsoft Blend, change the C1Scheduler.Theme property at design time:

1. Select the C1Scheduler control in your XAML window or page.
2. In the Properties panel, under **View**, click the drop-down arrow next to the C1Scheduler.Theme property and select **Reset**.
3. Click the C1Scheduler.Theme drop-down arrow again and choose one of the predefined themes.

To set the theme using the **ResourceID**, use the following XAML:

XAML

```
<c1:C1Scheduler x:Name="scheduler1" Theme="{DynamicResource
{ComponentResourceKey ResourceId=Office2007.Silver,
TypeInTargetAssembly={x:Type c1sched:C1Scheduler}}}"/>
```

To set the theme using **C1SchedulerResources** static fields, add the following code to your project:

Visual Basic

```
Scheduler1.Theme = C1SchedulerResources.Office2007Silver
```

C#

```
Scheduler1.Theme = C1SchedulerResources.Office2007Silver;
```

To set the theme by defining a **ResourceDictionary** and **DefaultThemeKey** in your Page, Window, or Application resources, use the following XAML:

XAML

```
<Page.Resources>
            <ResourceDictionary>
<ResourceDictionary x:Key="{x:Static my:C1Scheduler.DefaultThemeKey}"
Source="/C1.WPF.Schedule;component/themes/SchedulerThemes/Office2007/Blue.xaml" />
            </ResourceDictionary>
```

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

Note that this will affect all controls in the current scope.

You can also create your own theme **ResourceDictionaries** and use them with C1Scheduler.

The best way to customize one of the predefined themes is to include the default theme definition in a custom **ResourceDictionary** and redefine necessary resources, such as theme brushes, there.

Note: To ensure all default styles and templates continue to work correctly during customization, it is suggested that the resource keys are not changed from their default settings.

XAML

```
<Page x:Class="C1WPFSchedulerSamples.ThemedSchedulerWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:c1sched="clr-namespace:C1.WPF.Schedule;assembly=C1.WPF.Schedule"

xmlns:PresentationOptions="http://schemas.microsoft.com/winfx/2006/xaml/presentation/options"
      xmlns:sys="clr-namespace:System;assembly=mscorlib"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      mc:Ignorable="PresentationOptions"
      xmlns:local="clr-namespace:C1WPFSchedulerSamples"
      Title="Themed Scheduler"
      KeepAlive="True" Name="rootWindow">
      <Page.Resources>
            <ResourceDictionary>
                  <!--define theme resource dictionary -->
                  <ResourceDictionary x:Key="custom_theme">
                        <!-- include definition of default theme -->
                        <ResourceDictionary.MergedDictionaries>
                              <ResourceDictionary
            Source="/C1.WPF.Schedule;component/themes/SchedulerThemes/Office2007/Blue.xaml"
/>
                        </ResourceDictionary.MergedDictionaries>
                        <!-- redefine some resources -->
                        <Thickness x:Key="C1Scheduler_TimeBorder_Thickness">0</Thickness>
                        <sys:Boolean
x:Key="C1Scheduler_ShowNavigationPanels">True</sys:Boolean>
                        <Thickness x:Key="C1Scheduler_AllDayAreaBorder_Thickness">
1px,0,1px,2px</Thickness>
<SolidColorBrush x:Key="C1Scheduler_AlternateMonth_Brush" Color="#88FFFFFF"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_AllDayArea_Brush"
Color="#88FFFFFF"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_Background" Color="#FF5A8ECE"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_Border_Brush" Color="#FF000080"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_WorkHourBrush" Color="GhostWhite"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_WorkHourBorder_Brush"
Color="#98FFFFFF"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_WorkHourLightBorder_Brush"
```

```
Color="#C8FFFFFF" PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_FreeHour_Brush" Color="#D2FFFFFF"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_FreeHourBorder_Brush"
Color="#98FFFFFF"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_FreeHourLightBorder_Brush"
Color="#A8FFFFFF" PresentationOptions:Freeze="true"/>
                        <LinearGradientBrush x:Key="C1Scheduler_Day_SelectedBrush"
StartPoint="0,1"
EndPoint="1,0" PresentationOptions:Freeze="true">
                              <GradientStop Color="WhiteSmoke" Offset="0" />
                              <GradientStop Color="Navy" Offset="1" />
                        </LinearGradientBrush>
                        <LinearGradientBrush x:Key="C1Scheduler_TimeSlot_SelectedBrush"
StartPoint="0,0" EndPoint="1,0" PresentationOptions:Freeze="true">
                              <GradientStop Color="WhiteSmoke" Offset="0" />
                              <GradientStop Color="Navy" Offset="1" />
                        </LinearGradientBrush>
                        <SolidColorBrush x:Key="C1Scheduler_ControlArea_Brush"
Color="#FFE7EFFF"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_ControlAreaLines_Brush"
Color="#FF000080" PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_ControlAreaText_Brush"
Color="Navy"
PresentationOptions:Freeze="true"/>
                        <RadialGradientBrush x:Key="C1Scheduler_DayHeader_HoverBrush"
GradientOrigin="0.5,0.5" Center="0.5,0.5" RadiusX="0.5" RadiusY="1"
PresentationOptions:Freeze="true">
                              <GradientStop Color="#FFF79494" Offset="0" />
                              <GradientStop Color="Navy" Offset="1" />
                        </RadialGradientBrush>
                        <LinearGradientBrush x:Key="C1Scheduler_DayHeader_Brush"
EndPoint="0.5,1"
StartPoint="0.5,0" SpreadMethod="Pad"
MappingMode="RelativeToBoundingBox" PresentationOptions:Freeze="true">
                              <GradientStop Color="#FFB7C3D5" Offset="0"/>
                              <GradientStop Color="#FF000080" Offset="1"/>
                        </LinearGradientBrush>
                        <SolidColorBrush x:Key="C1Scheduler_DayHeaderText_Brush"
Color="White"
PresentationOptions:Freeze="true"/>
                        <SolidColorBrush x:Key="C1Scheduler_DayHeaderBorder_Brush"
Color="#00000000"
PresentationOptions:Freeze="true"/>
                        <LinearGradientBrush x:Key="C1Scheduler_AppointmentBgMask_Brush"
StartPoint="0,0" EndPoint="0,1" PresentationOptions:Freeze="true">
                              <GradientStop Color="#CCFFFFFF" Offset="0" />
                              <GradientStop Color="#10FFFFFF" Offset="1" />
                        </LinearGradientBrush>
                  </ResourceDictionary>
            </ResourceDictionary>
      </Page.Resources>
<Grid>
      <!-- set scheduler's theme to the custom_theme -->
```

```
      <c1:C1Scheduler Name="scheduler1" Theme="{StaticResource custom_theme}"
Background="CornflowerBlue" FontSize="12" FontWeight="Bold"/>
      </Grid>
</Page>
```

Notice that all of the brush definitions have **PresentationOptions:Freeze="true"**. This approach can be used for better performance.

Default C1Scheduler Theme Resources

The following table depicts the details of default C1Scheduler theme resources along with their keys:

Keys used by all default themes

Resource Key	Description
C1Scheduler_AllDayAreaBorder_Thickness	Determines the border thickness of All-Day area (used in a Day, Working Week and Office 2007 Week views).
C1Scheduler_AllDayArea_Brush	The brush which is used for coloring All-Day area background.
C1Scheduler_AllDayArea_SelectedBrush	The brush which is used for coloring All-Day area background for currently selected days.
C1Scheduler_AlternateMonthDay_Brush	The brush which is used for displaying background of alternate month days (used in the Month view).
C1Scheduler_AlternateMonthDayHeader_Brush	The brush which is used for displaying day header of alternate month days (used in the Month view).
C1Scheduler_AppointmentBgMask_Brush	The brush which is used as a mask for coloring Appointment background.
C1Scheduler_Border	The brush which is used as control's background if **C1Scheduler**.Background property has no local value.
C1Scheduler_Border_Brush	The brush which is used for coloring inter-day borders.
C1Scheduler_ControlArea_Brush	The brush which is used for coloring control area background (time ruler, etc.).
C1Scheduler_ControlAreaLines_Brush	The brush which is used for coloring control area borders (time ruler, etc.).
C1Scheduler_ControlAreaText_Brush	The brush which is used for coloring control area text (time ruler, etc.).
C1Scheduler_DayHeader_Brush	The brush which is used for coloring day headers.
C1Scheduler_DayHeader_HoverBrush	The brush which is used for coloring day headers when mouse is over.
C1Scheduler_DayHeaderBorder_Brush	The brush which is used for coloring day headers borders.
C1Scheduler_DayHeaderText_Brush	The brush which is used for coloring day headers text.

C1Scheduler_FreeHourBorder_Brush	The brush which is used for displaying free hours horizontal dark border.
C1Scheduler_FreeHour_Brush	The brush which is used for displaying background of work hours.
C1Scheduler_FreeHourLightBorder_Brush	The brush which is used for displaying free hours horizontal light border.
C1Scheduler_TimeSlot_SelectedBrush	The brush which is used for coloring selected time slot.
C1Scheduler_WorkHourBorder_Brush	The brush which is used for displaying working hours horizontal dark border.
C1Scheduler_WorkHour_Brush	The brush which is used for displaying background of work hours.
C1Scheduler_WorkHourLightBorder_Brush	The brush which is used for displaying working hours horizontal light border.

Office 2007 specific keys

Resource Key	Description
C1Scheduler_AllDayAreaTodayBorder _Thickness	Determines the border thickness of All-Day area (used in a Day, Working Week and Office 2007 Week views) for the current date.
C1Scheduler_AllDayAreaTodayBorder_Brush	Determines the border brush of All-Day area (used in a Day, Working Week and Office 2007 Week views) for the current date
C1Scheduler_NavPane_Brush	The brush which is used for coloring navigation panes.
C1Scheduler_NavPane_HoverBrush	The brush which is used for coloring navigation panes when mouse is over.
C1Scheduler_NavPaneBorder_Brush	The brush which is used for coloring navigation panes borders.
C1Scheduler_NavPaneText_Brush	The brush which is used for coloring navigation panes text.
C1Scheduler_ShowNavigationPanels	Determines whether control shows navigation panels for navigation to previous/next appointments.
C1Scheduler_Day_SelectedBrush	The brush which is used for coloring selected day background in a Month view.
C1Scheduler_TodayBorder_Brush	The brush wich is used for coloring the current day border.
C1Scheduler_TodayHeader_Brush	The brush which is used for coloring the current day header.
C1Scheduler_TodayHeader_HoverBrush	The brush which is used for coloring the current day header when mouse is over.
C1Scheduler_TodayHeaderBorder_Brush	The brush which is used for coloring the current day header border.
C1Scheduler_TodayHeaderText_Brush	The brush which is used for coloring the current day header text.
C1Scheduler_WeekTab_Brush	The brush which is used for coloring week tabs.
C1Scheduler_WeekTab_HoverBrush	The brush which is used for coloring week tabs when mouse is over.

C1Scheduler_WeekTabBorder_Brush	The brush which is used for coloring week tabs borders.
C1Scheduler_WeekTabText_Brush	The brush which is used for coloring week tabs text.

Note that C1Scheduler's **ThemeResources** properties for styles and templates are fully defined in XAML. When building your own styles and templates for C1Scheduler's UI, consider one of the following:

- using resources in the same way as in C1Scheduler's default styles and templates
- using your own logic for UI building

Tips for Creating Custom Styles and Templates

The following list includes tips for creating custom styles and templates:

- Keep the visual tree as simple as possible
- Setting **IsHittestVisible**="false" on some elements might improve performance
- Make sure that you don't do that on elements which have **CoverElementsPane.Orientaion** property set
- Use C1BrushBuilder as high in the visual tree as you can. The more C1BrushBuilders you use, the worse performance you have.

Creating a Custom Theme

A theme is a **ResourceDictionary** containing a collection of different resources such as Brushes and Styles and DataTemplates that target the WPF controls. The C1Scheduler and C1Calendar controls are composed of many smaller parts, each of which is created dynamically at run-time. All of these parts are described in a theme dictionary as a reusable resource. For example, there is only one Style definition for the Day Header which is used in all views.

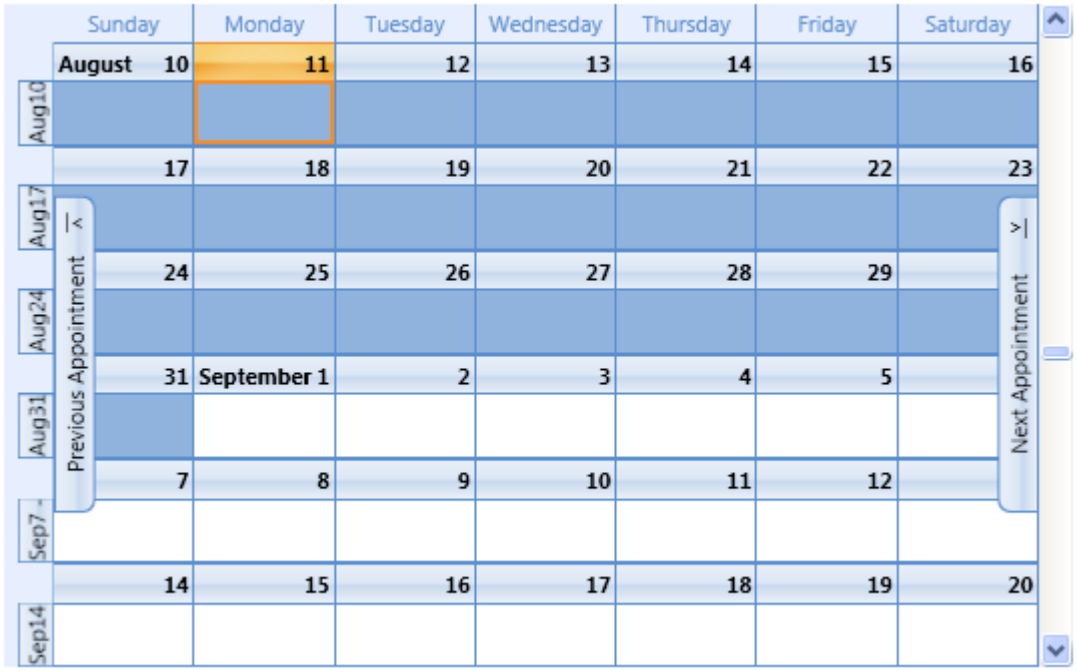
For brush resources, it is good practice to define every brush once, assign a unique key to it, and then use it later as a theme property for the styles and templates. Here is a XAML example of a brush defined and then used in a style setter and border background:

XAML

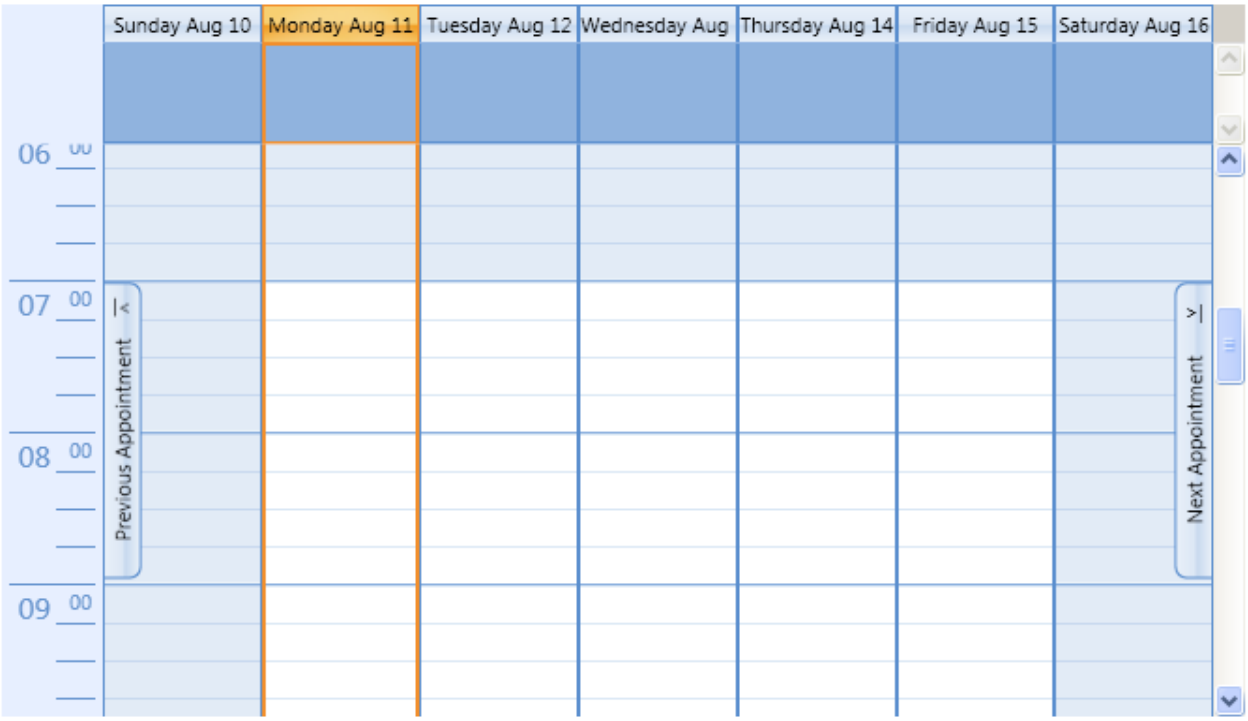
```
<!-- determines brush as frozen (for better performance) -->
<LinearGradientBrush x:Key="DayHeaderBrush" StartPoint="0,0" EndPoint="0,1"
PresentationOptions:Freeze="true">
   <GradientStop Color="#DDFFFFFF" Offset="0" />
   <GradientStop Color="#CAFFFFFF" Offset="0.5" />
   <GradientStop Color="#AAFFFFFF" Offset="0.6" />
   <GradientStop Color="#DDFFFFFF" Offset="1" />
</LinearGradientBrush>
<!-- use brush in style setter -->
<Setter Property="Background" Value="{Binding Path=Scheduler.Theme[DayHeaderBrush]}"
/>
<!-- use brush as border background -->
<Border Name="gradBrushRect"
Background="{Binding RelativeSource={RelativeSource AncestorType={x:Type
local:C1Scheduler}},
   Path=Theme[DayHeaderBrush]}" />
```

So, what are the "smaller parts" composing C1Scheduler? The VisualInterval is the key here. For the **Month View**, they are usually a one-day interval. For the **Day View**, they are the smallest time slot of several minutes. These intervals can be grouped into bigger parts. For example, in the Month view, intervals are grouped into weeks, and in the Day View, minutes are grouped into hours and hours are grouped into days. So, to determine the C1Scheduler appearance, we have to determine appearance of all these individual parts.

The next two images show how resource keys used in default themes correspond to UI representation. The first example is a schedule in **Month View**:



The second example is a schedule in **Week View**:



The first step at creating a new theme should be an understanding of how to split up the whole picture into smaller parts. Then you can create resources, styles, and templates for those smaller parts and use them for creating a new style for the whole control. Basically, if you keep the same resource keys which are used in the C1Scheduler source XAML, then you can re-template only some of the smaller parts which will be used automatically by default styles.

Creating the Theme Pack

To create your own theme pack, create a new Visual Studio 2008 project and include one or more theme ResourceDictionaries.

1. From the **File** menu in Visual Studio 2008, select **New Project**. The **New Project** dialog box appears.
2. Select **WPF User Control Library** from the list of Templates. Note that this option will appear for the **Windows** node of Visual Basic or C# **Project types**.
3. Enter a project location and name, **C1SchedulerThemePack**, for example, and click **OK**.
4. In the SolutionExplorer, right click the **UserControl1.xaml** file, select **Delete** from the menu, and click **OK**.
5. Add a reference to the C1.WPF.Schedule.dll assembly.
6. In the SolutionExplorer, right click the project name, click **Add | New Folder** and name this folder **themes**.
7. Right-click the **themes** folder and select **Add | Resource Dictionary**.
8. Enter **generic.xaml** in the **Name** text box and click **Add**.
9. Right-click the **themes** folder again and select **Add | Resource Dictionary**.
10. Enter **MyTheme.xaml** in the **Name** text box and click **Add**.
11. Open the **MyTheme.xaml** file and add namespace declarations inside the opening ResourceDictionary tag so it looks similar to the following:

XAML

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:C1.WPF.Schedule;assembly=C1.WPF.Schedule"
xmlns:storage="clr-namespace:C1.C1Schedule;assembly=C1.WPF.Schedule"
xmlns:PresentationOptions="http://schemas.microsoft.com/winfx/2006/xaml/presentation/options"
xmlns:sys="clr-namespace:System;assembly=mscorlib"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
```

```
mc:Ignorable="PresentationOptions">
```

12. If you want to redefine only part of the default theme, add the source theme definition to your **ResourceDictionary.MergedDictionaries** collection like in the following XAML:

XAML

```
<ResourceDictionary.MergedDictionaries>
<ResourceDictionary
Source="/C1.WPF.C1Schedule;component/themes/SchedulerThemes/Office2007/Blue.xaml" />
</ResourceDictionary.MergedDictionaries>
```

13. Add the brush definitions you want to redefine. Add the brushes definitions you want to redefine. For example, use the following XAML:

XAML

```
<SolidColorBrush x:Key="AlternateMonthBrush" Color="#08FFFFFF"
PresentationOptions:Freeze="true"/>
<SolidColorBrush x:Key="AllDayAreaBrush" Color="#11FFFFFF"
PresentationOptions:Freeze="true"/>
<SolidColorBrush x:Key="Background" Color="#FF111111" PresentationOptions:Freeze="true"/>
<SolidColorBrush x:Key="BorderBrush" Color="#000B0B0B" PresentationOptions:Freeze="true"/>
<SolidColorBrush x:Key="WorkHourBrush" Color="#FF303030" PresentationOptions:Freeze="true"/>
```

14. Override styles and templates defined in the included dictionary. For example, use the following XAML:

XAML

```
<!-- determines the style of the month grid header (week day names) -->
<Style x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler},
      ResourceId=MonthHeaderStyle}" TargetType="{x:Type ContentControl}">
      <Setter Property="Template">
            <Setter.Value>
                  <ControlTemplate TargetType="{x:Type ContentControl}">
                        <Grid>
                              <Border BorderThickness="0,0,1px,0"
BorderBrush="{Binding RelativeSource={RelativeSource
AncestorType={x:Type local:C1Scheduler}},
                                    Path=Theme[ControlAreaLinesBrush]}"
SnapsToDevicePixels="True"
Background="{Binding RelativeSource={RelativeSource
AncestorType={x:Type local:C1Scheduler}},
                                    Path=Theme[ControlAreaBrush]}" />
                              <ContentPresenter TextBlock.Foreground="Red" Margin="0,2,0,2"
                                    Content="{TemplateBinding Content}"
                                    HorizontalAlignment="Center" VerticalAlignment="Center"
/>
                              </Grid>
                        </ControlTemplate>
                  </Setter.Value>
            </Setter>
      </Style>
```

15. Save your changes. Note that this XAML will not work correctly until the ComponentResourceKey for the new theme is added to the **generic.xaml** file in the following steps.
16. Open **generic.xaml** file and add the following namespace declaration inside the opening ResourceDictionary tag: `xmlns:local="clr-namespace:C1.WPF.Schedule;assembly=C1.WPF.Schedule"`
17. Add the ComponentResourceKey for the new theme to the **generic.xaml** file using the following XAML:

XAML

```
<ResourceDictionary x:Key="{ComponentResourceKey
TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=MyTheme}"
Source="/C1SchedulerThemePack;component/themes/MyTheme.xaml" />
```

18. Save all changes and click select **Build | Build C1SchedulerThemePack** to build your theme assembly.

> **Tip:** If you want to create a theme with a number of colors, such as Office 2007 Blue, Silver and Black, place the full theme definition into a single file and use this for one color. Then add this file as a merged resource dictionary into the ResourceDictionaries for the other colors, and redefine the colors (Brush resources) only. Create ComponentResourceKeys for each color in the **generic.xaml** file.

Testing the Theme Pack

Once your first theme pack is created, make a test application for it.

1. Select **File | Add | New Project**.
2. Choose **WPF Application** from the list of **Templates**, specify a name, and click **OK**.
3. In the Solution Explorer, add a reference to the C1.WPF.Schedule.dll assembly to the new project you just added.
4. Also add a reference to the assembly that the **C1SchedulerThemePack** project generated here. It will be located in your **C1SchedulerThemePack** project folder, within **bin\Debug**: **C1SchedulerThemePack.dll**.
5. Open the **Window1.xaml** file in **Design** view and add a C1Scheduler control from the Toolbox.
6. Open the **Application.xaml** file and add a reference to the **generic.xaml** file from the **C1SchedulerThemePack** project to the application resources using the following XAML:

XAML

```
<Application.Resources>
      <ResourceDictionary
Source="/C1SchedulerThemePack;component/themes/generic.xaml" />
</Application.Resources>
```

7. Go back to the Window1.xaml file and change the C1Scheduler.Theme property of the C1Scheduler control to **MyTheme** using the following XAML. Delete grid tags.

XAML

```
<my:C1Scheduler Name="c1Scheduler1"
            Theme="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=my:C1Scheduler,
ResourceId=MyTheme}}"></my:C1Scheduler>
```

8. In the Solution Explorer, right-click the name of your new WPF application project, and choose **Set as Startup Project**.
9. Build the application and view the results in designer.

Creating a Theme in Blend

Note that Microsoft Expression Blend is not able to find external resource references at design time. However, if you are a designer and need to work in Blend, the steps below have been provided so that you can work with theme resources in the designer.

1. Create a **WPF Application (.exe)** in Blend.
2. If you want to redefine only part of the default C1Scheduler theme, copy all source XAML files with the theme definition into your application folder and add them to the project. Note that these files are installed with **Scheduler for WPF** to C:\Program Files\ComponentOne\WPF Edition\C1WPFNewSchedule\XAML\themes, by default. They may be in a different location if you installed the product elsewhere on your machine.
3. Create a new theme resource dictionary in the same project and use the source XAML as a merged resource dictionary within it.
4. Set the C1Scheduler.Theme property to your resource dictionary.
5. Edit your theme resource dictionary. You should be able to view the results in the designer.
6. After you have finished, if you want to share your theme between different applications, re-pack it into a separate assembly as was described previously.

C1Scheduler Silverlight Themes

C1Scheduler can be used along with C1.Silverlight.Theming themes. The only limitation of C1Scheduler comparing with other controls is the next:

You should explicitly set C1Scheduler.Theme property to the theme ResourceDictionary. For example:

C#

```
C1Theme theme = _new C1.Silverlight.Theming.WhistlerBlue.C1ThemeWhistlerBlue();
ResourceDictionary themeDictionary = C1Theme.GetCurrentThemeResources(theme);
sched1.Theme = themeDictionary;
```

Default C1Scheduler Theme Resources

The following table depicts the details of default C1Scheduler theme resources along with their keys:

Keys used by all default themes

Resource Key	Description
C1Scheduler_Background	The brush which is used as control's Background property default value.
C1Scheduler_Border_Brush	The brush which is used as control's BorderBrush property default value.
C1Scheduler_Border_Style	C1BrushBuilder style which is used for coloring borders of navigation panes, day headers and week tabs.
C1Scheduler_Foreground_Brush	The brush which is used as control's Foreground property default value.

C1Scheduler_Selected_Brush	The brush which is used as default value for C1Scheduler.SelectedBackground property. It is used for coloring selected day background in a Month view.
C1Scheduler_ControlArea_Brush	The brush which is used as default value for C1Scheduler.ControlBackground property. It is used for coloring control area background (time ruler, etc.).
C1Scheduler_ControlAreaText_Brush	The brush which is used as default value for C1Scheduler.ControlForeground property. It is used for coloring control area text and lines (time ruler, etc.).
C1Scheduler_NavPane_Brush	The brush which is used for coloring navigation panes.
C1Scheduler_NavPane_Style	C1BrushBuilder style which is used for coloring navigation panels.
C1Scheduler_NavPane_HoverBrush	The brush which is used for coloring navigation panes when mouse is over.
C1Scheduler_NavPane_HoverStyle	C1BrushBuilder style which is used for coloring navigation panels when mouse is over.
C1Scheduler_DayHeader_Brush	The brush which is used for coloring day headers.
C1Scheduler_DayHeader_Style	C1BrushBuilder style which is used for coloring day headers.
C1Scheduler_DayHeader_HoverBrush	The brush which is used for coloring day headers when mouse is over.
C1Scheduler_DayHeader_HoverStyle	C1BrushBuilder style which is used for coloring day headers when mouse is over.
C1Scheduler_Today_Brush	The brush which is used as default value for C1Scheduler.TodayBackground property. It is used as a base color for coloring the current day elements.
C1Scheduler_TodayHeader_Brush	The brush which is used for coloring the current day header.
C1Scheduler_TodayHeader_HoverBrush	The brush which is used for coloring the current day header when mouse is over.
C1Scheduler_AllDayAreaBorder_Thickness	Determines the border thickness of All-Day area (used in a Day, Working Week and Office 2007 Week views).
C1Scheduler_AllDayAreaTodayBorder _Thickness	Determines the border thickness of All-Day area (used in a Day, Working Week and Office 2007 Week views) for the current date.
C1Scheduler_AllDayArea_Brush	The brush which is used for coloring All-Day area background.
C1Scheduler_AllDayArea_SelectedBrush	The brush which is used for coloring All-Day area background for currently selected days.
C1Scheduler_AllDayArea_Style	C1BrushBuilder style which is used for coloring All-Day area.
C1Scheduler_WorkHour_Brush	The brush which is used for displaying background of work hours. It is used as default value for C1Scheduler. AlternatingBackground property.
C1Scheduler_WorkHourBorder_Brush	The brush which is used for displaying working hours horizontal dark border.
C1Scheduler_WorkHourLightBorder_Brush	The brush which is used for displaying working hours horizontal

	light border.
C1Scheduler_FreeHour_Brush	The brush which is used for displaying background of work hours.
C1Scheduler_FreeHourBorder_Brush	The brush which is used for displaying free hours horizontal dark border.
C1Scheduler_FreeHourLightBorder_Brush	The brush which is used for displaying free hours horizontal light border.
C1Scheduler_WeekTab_Brush	The brush which is used for coloring week tabs.
C1Scheduler_WeekTab_BrushStyle	C1BrushBuilder style which is used for coloring week tabs.
C1Scheduler_WeekTab_HoverBrush	The brush which is used for coloring week tabs when mouse is over.
C1Scheduler_WeekTab_HoverBrushStyle	C1BrushBuilder style which is used for coloring week tabs when mouse is over.
C1Scheduler_AlternateMonthDay_Brush	The brush which is used for displaying background of alternate month days (used in the Month view).
C1Scheduler_AppointmentBgMask_Brush	The brush which is used as a mask for coloring Appointment background.
C1Scheduler_AppointmentForeground_Brush	The brush which is used as default value for C1Scheduler.AppointmentForeground property. It is used for displaying appointment text.
C1Scheduler_TimeSlot_SelectedBrush	The brush which is used for coloring selected time slot.

Creating a Custom Theme

The easiest way to create custom theme for C1Scheduler is to inherit from one of default C1 themes and re-define some default resources as it is shown in MyCustomTheme sample (it can be found in Studio for Silverlight installation folder: ComponentOne\Studio for Silverlight\Samples\C1.Silverlight.Theming\CustomThemes\MyCustomTheme\).

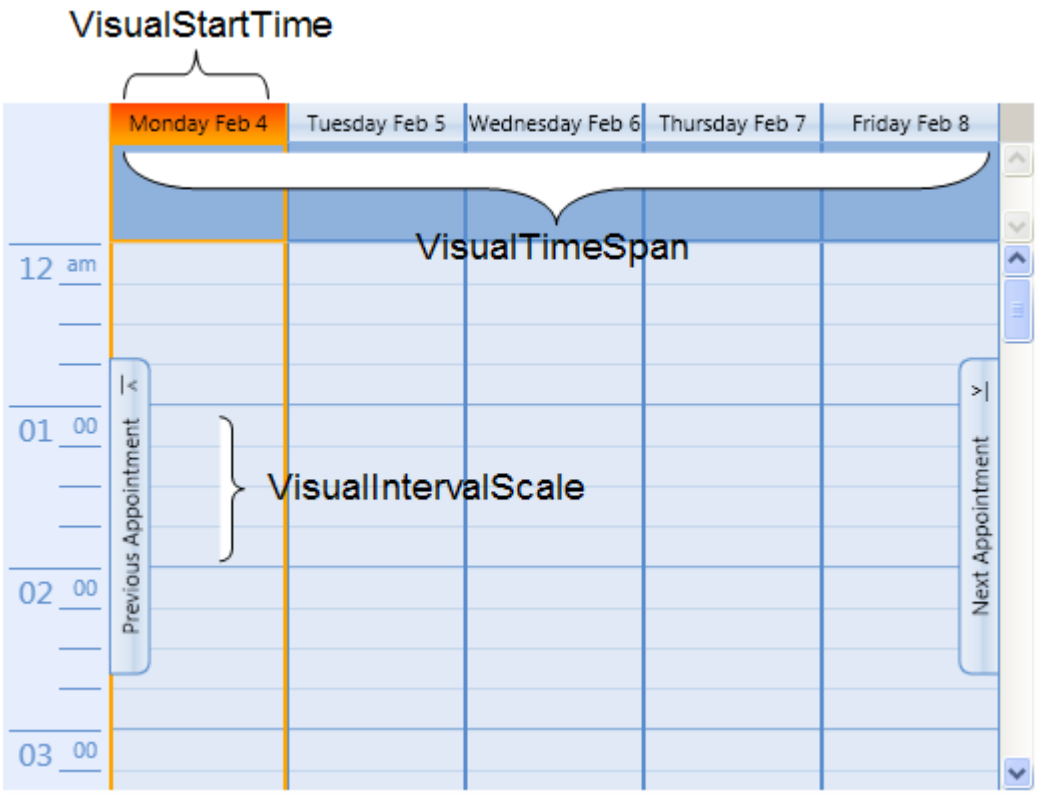
Customizing the User Interface

Scheduler for WPF and Silverlight provides a specific visual data model that is intended to help in creating a user interface (UI). The UI building concept for C1Scheduler is similar to a grouped **System.Windows.Controls.ItemsControl** with a special **ItemsControl.ItemsSource** collection.

For example, the C1Scheduler.VisualStartTime sets the first date to appear in the schedule.
The C1Scheduler.VisualTimeSpan property defines the amount of time, or in most cases, the number of days shown in a view. The C1Scheduler.VisualIntervalScale property breaks the C1Scheduler.VisualTimeSpan even further into increments. To preserve custom scaling in the views, C1Scheduler class offers **C1Scheduler.SmallVisualIntervalScale** property. Refer to the topic Customizing Time Span for Different Views to know how you can preserve custom scaling while switching from one view to another.

Using the *Working Week View* view style as an example, the C1Scheduler.VisualStartTime is the first Monday of the current week, by default. The C1Scheduler.VisualTimeSpan is 5:00:00:00, or five days, Monday through Friday which is

a typical work week. The C1Scheduler.VisualIntervalScale is 00:15:00 which means that each hour of the schedule is shown broken down into 15 minute intervals.



Note: The C1Scheduler.VisualStartTime property value is determined automatically by the C1Scheduler.VisualTimeSpan and C1Scheduler.SelectedDateTime properties, so it should not be set in most cases; however, the C1Scheduler.VisualTimeSpan and C1Scheduler.SelectedDateTime properties must be set. This is important for week and month views where the current date is not always the first date to appear in the schedule.

For specified C1Scheduler.VisualStartTime, C1Scheduler.VisualTimeSpan, and C1Scheduler.VisualIntervalScale properties, the VisualIntervalCollection is filled with VisualInterval objects, each defining a period of time of the C1Scheduler.VisualIntervalScale length.

The items in the VisualIntervalCollection can be grouped; grouping criteria is defined in the **VisualIntervalGroupDescriptions** collection, which contains one item per grouping level. A UI for each group level, which consists of a Header, a Panel containing group items, the group item's UI, and a layout that gathers these parts together, is defined in the **VisualIntervalGroupStyles** collection, which contains one item per grouping level.

To define the VisualInterval's UI representation, the C1Scheduler.VisualIntervalTemplate property should be used. The C1Scheduler.VisualIntervalPanel property defines a Panel where the VisualInterval's items are placed.

To provide a group view for the VisualIntervalCollection, the **C1Scheduler.VisualIntervalsView** property should be used, which is a **System.Windows.Data.PagedCollectionView** derived object that has VisualInterval as the SourceCollection.

A **DataContext** for the VisualInterval's UI DataTemplate is the VisualInterval itself, so the construction {Binding property_name} in the DataTemplate XAML code means binding to the property_name property of VisualInterval.

A **DataContext** for the group item's UI DataTemplate is a VisualIntervalGroup object, which is derived from VisualInterval. A time period that is represented by this VisualIntervalGroup is a union of periods of child items (child groups or intervals). The only properties that it adds to the base class are:

- Group of type **System.Windows.Data.CollectionViewGroup**; this is a data object that represents a group item in the **VisualIntervalCollectionView.Groups** collection.
- VisualIntervals of type **C1.WPF.C1Schedule.VisualIntervalCollection** (a collection of child VisualInterval objects).

The VisualInterval class, among others, contains the **Appointments** collection that represents the appointments that intersect with the interval. This collection doesn't contain **Appointment** instances directly; instead it holds the **IntervalAppointment** class instances, which in turn has a reference to an appointment they represent (the Appointment property), along with some helper properties for indication of whether this appointment starts in this interval, whether it's finished here, and so on.

When the **ControlTemplate** for C1Scheduler is being created (**C1Scheduler.Template**), the C1SchedulerPresenter object should be used as a markup for a place where intervals and their groups will be shown.

For example, in order to define an Outlook-style Work Week view with 30-minute time intervals, you may use the following code:

Visual Basic

```
C1Scheduler1.ChangeStyle(C1Scheduler1.WorkingWeekStyle)
C1Scheduler1.VisualIntervalScale = TimeSpan.FromMinutes(30)
C1Scheduler1.VisualTimeSpan = TimeSpan.FromDays(5)
Group Level1
PropertyNames = "StartTime.Day" (group by VisualInterval.StartTime.Day)
GroupStyle = (Header: day name; Panel : some Panel with Horizontal orientation)
Group Level2
PropertyNames = "StartTime.Hour" (group by VisualInterval.StartTime.Hour)
GroupStyle = (Header: none; Panel : some Panel with Vertical orientation with a
Border around)
```

VisualIntervalTemplate = something that reacts on double-click and calls a Command that brings an appointment creation dialog (NewAppointmentDialogCommand, see below).

The following XAML is an example of how to define groups:

XAML

```
<!-- Group descriptions -->
<Setter Property="c1sched:C1Scheduler.VisualIntervalGroupDescriptions">
 <Setter.Value>
   <c1sched:IntervalGroupDescriptionCollection>
     <c1sched:VisualIntervalGroupDescription PropertyName="StartTime.Day" />
     <c1sched:VisualIntervalGroupDescription PropertyName="StartTime.Hour" />
   </c1sched:IntervalGroupDescriptionCollection>
 </Setter.Value>
</Setter>

<!-- Group styles definition -->
<Setter Property="VisualIntervalGroupStyles">
 <Setter.Value>
   <c1sched:IntervalGroupStyleCollection>
      <c1sched:GroupStyle ContainerStyleSelector="{StaticResource
DayGroupStyleSelector}">
         <c1sched:GroupStyle.Panel>
```

```
            <DataTemplate>
               <c1:C1UniformGrid Rows="1" />
            </DataTemplate>
         </c1sched:GroupStyle.Panel>
      </c1sched:GroupStyle>
      <c1sched:GroupStyle ContainerStyleSelector="{StaticResource
TimeSlotGroupStyleSelector}">
         <c1sched:GroupStyle.Panel>
            <DataTemplate>
              <c1:C1UniformGrid Rows="24" Columns="1" />
            </DataTemplate>
         </c1sched:GroupStyle.Panel>
       </c1sched:GroupStyle>
   </c1sched:IntervalGroupStyleCollection>
 </Setter.Value>
</Setter>
```

To provide a custom look for the C1Scheduler control:

1. To define a general layout model for a C1Scheduler control, the **C1Scheduler**.Template property should be assigned. This is usually done through the Setter property of a Style. The template may contain any UI elements, but in some places of the template's visual tree, the following placeholder elements should appear:

    - **VisualIntervalGroupsPresenter** - to designate a place where the collection of VisualIntervalGroup objects will appear;
    - **AppointmentsCoverPane** – to provide a surface for drawing appointment boxes;
    - **C1SchedulerPresenter** – to define a place where a pane representing schedule time intervals will appear.

    Note that each of the placeholders enumerated above is optional.

2. To define grouping:
    - Set VisualIntervalGroupDescriptions to define grouping criteria applied to the items of the VisualIntervalCollection;
    - Set C1Scheduler.VisualIntervalGroupStyles to define a UI for each group level;
3. To define a layout of elements representing VisualInterval objects from the VisualIntervalCollection, set the C1Scheduler.VisualIntervalPanel property.
4. To define the VisualInterval representation, either set the C1Scheduler.VisualIntervalTemplate property or use the TimeSlotTemplateSelector.

Simplifying User Interface Creation

In order to simplify the building of a user interface based on C1Scheduler's visual data model, the AutoDistributionGrid grid has been created. It is derived from the **System.Windows.Controls.Grid** control, but it provides additional functionality. For example:

1. Child elements are scattered by rows or columns depending on the **Orientation** property value (**Horizontal** or **Vertical** respectively).
2. Settable and bindable RowCount and ColumnCount properties, for example the **DependencyProperty**, allows you to define row and column counts numerically, without adding or removing items in the **Grid.RowDefinitions/ColumnDefinitions** collections. Along with the bindable VisualChildCount property, it provides the ability to have as many rows as children.

3. You can define grid specific characteristics, such as position and span, for certain child elements. AutoDistributionGrid has a ChildrenDistributionInfo collection of DistributionInfo objects; each DistributionInfo object instructs the grid's child object at index **ElementIndex** to be (optionally) placed in a cell with **Row** and **Column** indexes and to have **RowSpan** and **ColumnSpan** spans. If the position is redefined for a certain element, then the next element will be placed in a cell next to this element according to the **Orientation** specified. If a span is defined and the span direction conforms to the **Orientation**, then the next element will skip over the span. Each DistributionInfo item's information can be propagated to a number (fixed or infinite) of next elements, which is specified in the **Propagate** property.

Showing Multiple Day Appointments

The AppointmentsCoverPane control visually represents a set of appointments that fit in a time range exposed by a current view, and to draw appointment boxes relative to user interface (UI) elements representing **C1Scheduler.VisualIntervals** covered by the appointment. This control, when placed somewhere inside a C1Scheduler visual tree, usually in the C1Scheduler's **ControlTemplate**, provides a surface where appointment boxes are drawn relative to the UI for **C1Scheduler.VisualIntervals**. The AppointmentsCoverPane control is able to recognize the case when an appointment box must be divided into two or more visual boxes. For example, in Month view, if an appointment covers three days, AppointmentsCoverPane will automatically draw three boxes in each corresponding day of the appointment.

The content of an AppointmentsCoverPane appointment box is represented by the DataTemplate defined in the C1Scheduler.IntervalAppointmentTemplate property.

The AppointmentsCoverPane control provides functionality for an arbitrary UI representing **C1Scheduler.VisualIntervals**. To make this possible, each element that can be treated as a VisualInterval UI representative, usually an outer (root) element in the C1Scheduler.VisualIntervalTemplate definition, and must have the attached CoverElementsPane.OrientationProperty field assigned. Note that CoverElementsPane is the base class for the AppointmentsCoverPane class. The assigned value indicates a chronological flow direction of interval elements and can take **Horizontal** or **Vertical** values. For example, interval elements in the **Working Week View** will have it assigned to **Vertical**, while elements of Month View assign it to **Horizontal**.

The C1Scheduler visual tree can contain several **AppointmentsCoverPanes**. For example, the default DayView template contains one AppointmentsCoverPane to display all-day events in day headers and the second AppointmentsCoverPane to display short appointments over the time slots. To filter appointments which should be displayed by the AppointmentsCoverPane, assign the AppointmentsCoverPane.Appointmentfilter attached property and the CoverElementsPane.PaneName attached property to the C1Scheduler.VisualIntervalTemplate definition. The AppointmentsCoverPane.Appointmentfilter attached property can be set to **AppointmentFilterEnum.Event** (show only all-day and multiple-day appointments), **AppointmentFilterEnum.Appointment** (show only appointments with a duration of 24 hours or less) and **AppointmentFilterEnum.All** values. The CoverElementsPane.PaneName property should be set to the name of the AppointmentsCoverPane object that should display an appointment. For example:

Silverlight

XAML

```
<Setter Property="c1sched:C1Scheduler.VisualIntervalTemplate">
 <Setter.Value>
   <DataTemplate>
      <Border Background="{Binding Path=StatusBrush}" Opacity="0.3"
              c1sched:AppointmentsCoverPane.AppointmentFilter="Appointment"
              c1sched:CoverElementsPane.Orientation="Vertical"
              c1sched:CoverElementsPane.PaneName="appPane" MinHeight="20"/>
   </DataTemplate>
 </Setter.Value>
</Setter>
```

WPF

XAML

```
<Setter Property="local:C1Scheduler.VisualIntervalTemplate">
    <Setter.Value>
      <DataTemplate>
        <Border x:Name="IntervalBorder" SnapsToDevicePixels="True"
              Background="{Binding Path=Scheduler.Theme.WorkHourBrush}"
                    BorderBrush="{Binding
Path=Scheduler.Theme.WorkHourLightBorderBrush}"
                    BorderThickness="0,1px,0,0"
                    local:AppointmentsCoverPane.AppointmentFilter="Appointment"
                    local:CoverElementsPane.Orientation="Vertical"
                    local:CoverElementsPane.PaneName="appPane"
                    MinHeight="20">
          <Border.InputBindings>
            <MouseBinding MouseAction="LeftDoubleClick"
                Command="local:C1Scheduler.NewAppointmentDialogCommand"/>
          </Border.InputBindings>
        </Border>
        <DataTemplate.Triggers>
          <DataTrigger Binding="{Binding Path=IsSelected}" Value="True">
            <Setter TargetName="IntervalBorder" Property="Background"
                    Value="{Binding Path=Scheduler.Theme.SelectedSlotBrush}" />
          </DataTrigger>
        </DataTemplate.Triggers>
      </DataTemplate>
    </Setter.Value>
  </Setter>
```

Selecting Styles and Templates for VisualIntervals

In order to provide a fast and easy way for selecting appropriate styles and templates for **C1Scheduler.VisualIntervals**, the C1Scheduler control provides five classes: TimeSlotGroupStyleSelector, DayGroupStyleSelector, TimeSlotTemplateSelector, DayIntervalStyleSelector and TimeSlotStyleSelector.

The TimeSlotGroupStyleSelector class provides a way to apply time slot group styles for working and free hours in **Day View** and **Working Week View** modes. To use it:

1. Create an instance of the class:
   `<local:TimeSlotGroupStyleSelector x:Key="TimeSlotGroupStyleSelector"/>`
2. Define two group styles in the TimeSlotGroupStyleSelector resources:
   - with the key "PART_WorkHourStyle" for work hours;
   - with the key "PART_FreeHourStyle" for free hours.
3. Specify a style selector in the group definition:
   `<GroupStyle ContainerStyleSelector="{StaticResource TimeSlotGroupStyleSelector}">`

The **DayGroupStylesSelector** class provides a way to apply day group styles for an Office 2003/2007 look in Day View and Working Week View modes. To use it:

1. Create an instance of the class:
   ```
   <local:DayGroupStyleSelector x:Key="DayGroupStyleSelector"/>
   ```
2. Define two group styles in the **DayGroupStylesSelector resources**:
   - with the key "PART_Day2003Style" for the Office 2003 look;
   - with the key "PART_Day2007Style" for the Office 2007 look.
3. Specify a style selector in the group definition:
   ```
   <GroupStyle ContainerStyleSelector="{StaticResource DayGroupStyleSelector}">
   ```

The TimeSlotTemplateSelector class provides a way to choose a **DataTemplate** for the VisualInterval object representing the single time slot in **Day View** and **Working Week View** modes. To use it:

1. Create an instance of the class:
   ```
   <local:TimeSlotTemplateSelector x:Key="TimeSlotTemplateSelector"/>
   ```
2. Define two DataTemplates in the **TimeSlotTemplateSelector resources**:
   - with the key "PART_FreeSlotTemplate" for free time;
   - with the key "PART_WorkSlotTemplate" for working time.
3. Specify an ItemTemplateSelector for the C1SchedulerPresenter object:
   ```
   <local:C1SchedulerPresenter ItemTemplateSelector="{StaticResource
   TimeSlotTemplateSelector}" />
   ```

   Note that if you use this method for choosing a VisualInterval DataTemplate, you shouldn't set the **C1Scheduler.VisualIntervalTemplate** property.

The DayIntervalStyleSelector class provides a way to apply day group styles for ordinal/current days. To use it:

1. Create an instance of the class:
   ```
   <local:DayIntervalStyleSelector x:Key=”DayIntervalStyleSelector"/>
   ```
2. Define two group styles in the DayIntervalStyleSelector resources:
   - with the key "C1Scheduler_Day_Style" for ordinal days;
   - with the key "C1Scheduler_Today_Style" the current date.
3. Specify a style selector in the group definition:
   ```
   <GroupStyle ContainerStyleSelector="{StaticResource DayIntervalStyleSelector}">
   ```

The TimeSlotStyleSelector class provides a way to apply time slot styles for working and free hours in **Day View** and **Working Week View** modes. To use it:

1. ```
   Create an instance of the class:
       <local:TimeSlotStyleSelector x:Key="TimeSlotStyleSelector"/>
   ```
2. Define two group styles in the TimeSlotStyleSelector resources:
   - with the key "C1Scheduler_WorkSlot_Style" for work hours;
   - with the key "C1Scheduler_FreeSlot_Style" for free hours.
3. Specify an **ItemContainerStyleSelector** for the C1SchedulerPresenter object:
   ```
   <local:C1SchedulerPresenter ItemContainerStyleSelector="{StaticResource
   TimeSlotTemplateSelector}" />
   ```

C1Scheduler Commands

In order to offer the ability to use some types of C1Scheduler functionality declaratively in XAML, C1Scheduler provides a number of commands that can be issued with the help of instances of the **ButtonBase** derived classes.

The following table describes the **Navigation** commands:
   ```

Navigation Commands (an example of a sender of these commands is the C1SchedulerScrollBar control)	
DecrementStartTimeSmallCommand	Decrements the C1Scheduler.VisualStartTime property value on the amount specified in the C1Scheduler.SmallStartTimeChange property.
DecrementStartTimeLargeCommand	Decrements the C1Scheduler.VisualStartTime property value on the amount specified in the C1Scheduler.LargeStartTimeChange property.
IncrementStartTimeSmallCommand	Increments the C1Scheduler.VisualStartTime property value on the amount specified in the C1Scheduler.SmallStartTimeChange property.
IncrementStartTimeLargeCommand	Increments the C1Scheduler.VisualStartTime property value on the amount specified in the C1Scheduler.LargeStartTimeChange property.
SetRelativeStartTimeCommand	Sets the C1Scheduler.VisualStartTime property to a value between the Start and End property values based on the specified coefficient. Command parameter: a floating point number in a range between 0 and 1 or its string representation.
NavigateToPreviousAppointmentCommand	Set focus to the nearest appointment before the C1Scheduler.SelectedDateTime is visible in the control UI.
NavigateToNextAppointmentCommand	Sets focus to the nearest appointment after the C1Scheduler.SelectedDateTime is visible in the control UI.

The following table describes the **Dialog** commands:

Dialog Commands	
EditAppointmentDialogCommand	Opens the **Edit Appointment** dialog box for an existing appointment. Command parameter is one of the next values: • the Appointment to edit; • IList of Appointment objects to edit; • IList of Reminder objects whose parent appointments should be edited. If a parameter value is not specified, then the control will try to get an appointment from the sender's DataContext.
NewAppointmentDialogCommand	Adds a new appointment and opens the **Edit Appointment** dialog box for it. The time interval for which an appointment will be created is determined by the sender's DataContext.
EditRecurrenceDialogCommand	Opens the **Edit Recurrence** dialog box for an appointment's recurrence. Command parameter: the Appointment whose RecurrencePattern will be edited. If a parameter value is not specified, then the control will try to get an appointment from the sender's DataContext.

SelectFromListDialogCommand	Opens the Select Resources/Categories/Contacts, and so on dialog box. Command parameter should be an array from 4 or 5 values: 1. The master list to choose from. For example ResourceStorage.Resources. 2. The resulting list where to put selected items. For example, the Appointment.Resources list. 3. The System.Type value specifying the type of items in both lists. 4. The reference to the owning window if any. 5. The String value to show as the dialog window title. This parameter is optional.

The following table describes the **Import/Export** commands:

Import/Export Commands	
ImportCommand	Imports C1Scheduler data from the file. This command opens the "Open File" dialog box and then tries to import data from the selected file.
ExportCommand	Exports C1Scheduler data to the file. This command opens "Save File" dialog and then exports data to file. Command parameter is one of the next values: • the Appointment for saving; • Ilist<Appointment> for saving; • null – to export all Scheduler data.

The following table describes the **Reminder** commands:

Reminder Commands	
DismissCommand	Dismisses reminders. Command parameter is one of the next values: • IList of Reminder objects to dismiss; • the Reminder object to dismiss. If a command parameter is not specified, all active reminders will be dismissed.
SnoozeCommand	Snoozes reminders. Command parameter may contain an array of 2 values: 1. The TimeSpan value specifying time interval used for

	snoozing. 2. One of the next values: • IList of Reminder objects to snooze; • the Reminder object to snooze. This item is optional. If it is not specified, all active reminders will be snoozed.

The following table describes some additional commands:

Other Commands	
DeleteSelectedAppointmentCommand	Deletes an appointment which is currently selected in the UI and referenced in the C1Scheduler.SelectedAppointment property.
ChangeStyleCommand	Changes the Style property with the specified Style or a style referenced by the specified ResourceDictionary key. Command parameter: Style or a ResourceDictionary key representing a Style.
NavigateToPreviousGroupCommand	Defines the command that navigates C1Scheduler UI to the previous SchedulerGroupItem object. Optional command parameter is one of the next values: • "Page" for the page navigation. The page size is determined by the GroupPageSize property value. • "Home" for navigating to the first group.
NavigateToNextGroupCommand	Defines the command that navigates C1Scheduler UI to the next SchedulerGroupItem object. Optional command parameter is one of the next values: • "Page" for the page navigation. The page size is determined by the GroupPageSize property value • "End" for navigating to the last group.
HideGroupCommand	Defines the command that hides the specified SchedulerGroupItem object. Command parameter must specify the SchedulerGroupItem object to hide.

Assigning Values to a Nested Property

XAML does not provide the ability to assign a value to a nested property. However, the object model of the C1Scheduler.DataStorage contains nested properties by nature. To work around this limitation, the NestedPropertySetter class can be used.

Note: The NestedPropertySetter class works in a conjunction with C1Scheduler only.

Elements of this class, being placed as children of a C1Scheduler element in XAML, represent setters as **Property**/**Value** pairs where the **Property** specifies a property path relative to a parent C1Scheduler element. In the following example, the **C1BindingSource.DataMember** property is assigned to the **Appointments** table of a database. The **C1BindingSource.DataSource** is set to the data set resource in the project:

XAML

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

Binding C1Scheduler using WPF

The following topics walk you through the steps of binding the C1Scheduler control to a data source and using the PropertyBridge class for binding non-dependency properties.

Binding C1Scheduler to a Data Source

In this topic, we will bind to C1Scheduler the Schedule.mdb database that is installed with **Scheduler for WPF** by default. First we must configure the datasource. Then we can add the dataset as a resource in the Blend project and map to the C1Scheduler Data Storage.

To configure the data source in Visual Studio 2008:

1. Create a new WPF project in Microsoft Visual Studio:
    a. Select File | New Project.
    b. Expand the **Visual C#** node and select **NET Framework 3.0** (or later). Note that you may have to expand the **Other Languages** node to find **Visual C#**.
    c. Select **WPF Application** from the *Templates* pane.
    d. Enter a name for the project in the **Name** text box and click **OK**. A new project is created.
2. From the **Project** menu, select **Add Page**. The **Add New Item** window appears. **Visual C#** will be selected under *Categories*, and **Page (WPF)** will be selected in the *Templates* pane.
3. Leave **Page1.xaml** in the **Name** text box and click **Add**.
4. If it is not already open, double-click **Page1.xaml.cs** under the **Page1.xaml** node in the Solution Explorer to open it.
5. Add a reference to C1.WPF.Schedule:
    a. Select Project | Add Reference.
    b. Browse to find the location of the assembly file. When **Scheduler for WPF** is installed, this file is installed to C:\Program Files\ComponentOne\WPF Edition, by default. Note that the location may be different if you performed a custom install.
    c. Add the following using statements to the page:

XAML

```
using C1.WPF.Schedule;
```

```
using C1.Schedule;
using MYProjectNAME.ScheduleDataSetTableAdapters;
```

Note that this last using statement should contain the name of your project to work correctly. It will be used to set up the table adapter for your data set.

6. Next we will add the data source we are binding to:
   a. Select **Data | Add New Data Source**. The **Data Source Configuration Wizard** appears.
   b. Select **Database** and click **Next**.
   c. Click the **New Connection** button and browse to locate your database. In this example we will use the Schedule.mdb database installed with **Scheduler for WPF**. This file was placed in the Common folder during installation.
   d. Click **Next** once you create the new connection. It is not necessary to copy this file to your project, so click **No** if you receive a dialog box asking you to do this. The connection is given a name.
   e. Click **Next** to save the connection string.
   f. Select the database objects to use in the dataset and click **Finish**.
7. Add the following C# code to your Page1.xaml.cs **Page1** class so it looks like the following. This code will use the data from the dataset to fill the schedule.

XAML

```
public partial class Page1 : System.Windows.Controls.Page
    {
        private ScheduleDataSet _schedulerDataSet = null;

        public Page1()
        {
            InitializeComponent();
            // Use the data set to fill in the data.
            FillData(SchedulerDataSet);
        }

        public ScheduleDataSet SchedulerDataSet
        {
            get
            {
                if (_schedulerDataSet == null)
                {
                    _schedulerDataSet = Resources["dataSet"] as ScheduleDataSet;
                }
                return _schedulerDataSet;
            }
        }

        private void FillData(ScheduleDataSet ds)
        {
            if (ds == null)
                return;
            AppointmentsTableAdapter appAd = new AppointmentsTableAdapter();
            appAd.Fill(ds.Appointments);
        }

    }
```

8. Save and close the project.

To add the new dataset as a resource to your Blend project:

1. In Blend, open the project (.sln) that you created in Visual Studio 2008.
2. Double-click the **Page1.xaml** in the **Project** panel to open the page and click the **Design** tab to access the **Design** view, if necessary.
3. Add a C1Scheduler control to the page. See Adding the **Scheduler for WPF** Controls to a Blend Project for more information on how to do this.
4. Click the **XAML** tab to switch to **XAML** view.
5. Create a CLR namespace by adding the following XAML code to the namespace list within the **Page** tag. Use your project's name for the clr-namespace value.

   xmlns:local="clr-namespace:MYProjectNAME"

   Your XAML code should look similar to the following:

   XAML

```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="WpfApplication1.Page1"
      xmlns:local="clr-namespace:WpfApplication1"
    Title="Page1" xmlns:c1sched="http://schemas.componentone.com/wpf/Schedule">
```

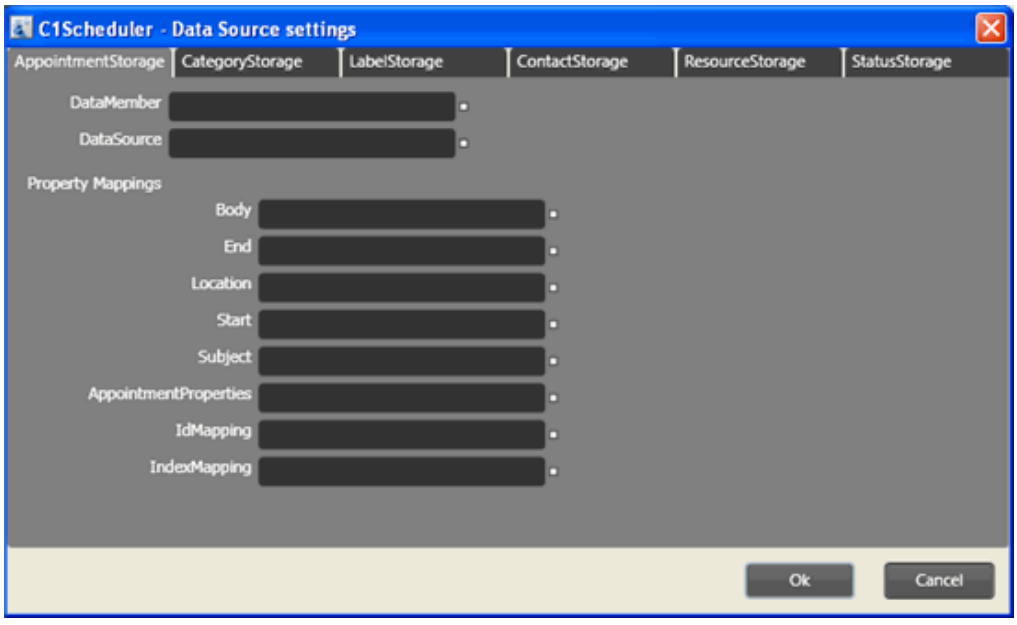
6. Select **Project | Build Solution**.
7. Add the ScheduleDataSet as a resource by entering the following code after the opening **<Page>** tag:

   XAML

```
<Page.Resources>
    <local:ScheduleDataSet x:Key="dataSet" />
</Page.Resources>
```

To map to the C1Scheduler Data Storage:

1. In your Blend project, click the **Design** tab to switch back to **Design** view of Page1.xaml.
2. Select the C1Scheduler control on **Page1.xaml** of your project.
3. In the **Properties** panel, under **Data**, click the **DataStorage** button. The **Data Source settings** dialog box appears.



4. Click the **Advanced Property Options** button next to the **DataSource** property, select **Local Resource** and check **dataSet**. This property is set to the ScheduleDataSet.
5. Enter **Appointments** next to the **DataMember** property. C1Scheduler will map to the **Appointments** table and use its data to fill in the schedule.
6. Next, set the **mappings** for the properties to the corresponding data fields in the **Appointments** table. Enter the following text for each of the **Property Mappings** items:

Property	Text
Body	Body
End	End
Location	Location
Start	Start
Subject	Subject
AppointmentProperties	Properties
IdMapping	Id
IndexMapping	N/A

7. Click **OK** to close the **Property Dialog Editor**. The database is now mapped to the **Appointment Storage**. The XAML code for the mappings looks like the following:

XAML

```
<c1:C1Scheduler Margin="0,0,-80,-80" Theme="{DynamicResource {ComponentResourceKey
ResourceId=Office2007.Blue, TypeInTargetAssembly={x:Type c1sched:C1Scheduler}}}">
      <c1:NestedPropertySetter PropertyName="DataStorage.AppointmentStorage.DataMember"
Value="Appointments"/>
      <c1:NestedPropertySetter PropertyName="DataStorage.AppointmentStorage.DataSource"
Value="{DynamicResource dataSet}"/>
      <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"/>
      <c1:C1Scheduler>
```

Data Binding Using the PropertyBridge Class

The PropertyBridge class exposes two dependency properties, **PropertyBridge.Source** and **PropertyBridge.Target**, of the **System.Obje**ct type, and keeps these property values equal. In other words, when the value of one property is changed, the other property is set to the same value. This simple behavior allows you to use non-DependencyProperty properties along with WPF mechanisms that are designed to work with DependencyProperty-only properties. The following examples show how the PropertyBridge class can be used:

- **Binding between two non-dependency properties:**
  Assign the **PropertyBridge.Source** property with a TwoWay binding having one non-dependency property as a source, and assign the **PropertyBridge.Target** property with a TwoWay binding having another non-dependency property as a source. Then the non-dependency properties will behave as bound properties. This will work well only in classes where exposing these non-dependency properties supports the **INotifyPropertyChanged** interfaces, which is the case for most classes from the C1ScheduleStorage object model.
- **Setting a non-dependency property value from with a Trigger:**
  Assign the **PropertyBridge.Source** property with a **TwoWay** or **OneWayToSource** binding having one non-dependency property as a source, and using Trigger's Setter to set a value to the **PropertyBridge.Target** property of PropertyBridge – this value will be assigned to the non-dependency property which is bound to **PropertyBridge.Source**.
- **Many-to-many binding:**
  Assign **PropertyBridge.Source** and **PropertyBridge.Target** with MultiBinding bindings to get many-to-many binding.
- **Assign a value to a nested property:**
  Set the target to a **TwoWay** or **OneWayToSource** binding with a Path referencing a nested property. Then, assign **PropertyBridge.Source** (directly or from within a Setter). The nested property will be assigned to this value.
- **Assign the property of an object that is not accessible directly:**
  Similar to assigning a value to a nested property, using **RelativeSource** in binding to PropertyBridge.Target, assign a property value for an element that can't be referenced directly in XAML, for example, **TemplatedParent** or some parent element in the visual tree.

The PropertyBridge class is derived from **FrameworkElement** and in order to work properly, it should be placed somewhere in the visual tree among other elements that it should communicate with. The derivation from **FrameworkElement** is intentional; it allows the PropertyBridge to be a part of a visual tree, which in turn provides bindings established on its properties with the correct context. For example, another theoretical option is to have PropertyBridge in the **ResourceDictionary**, but in this case, the bindings would be inoperable.

The **PropertyBridge.Visibility** property is set to **Collapsed** by default, so this object will not appear on a screen and doesn't participate in layout measurement and arrangement processes; it doesn't corrupt a visual representation of the visual tree where it is placed.

Binding C1Scheduler using Silverlight

The following topics walk you through the steps of binding the C1Scheduler control to a data source.

C1Scheduler supports binding to custom data sources. The following are some examples of using a custom data source:

- an instance of the C1.Silverlight.Data.DataSet, C1.Silverlight.Data.DataView or C1.Silverlight.Data.DataTable class. Two-way binding is supported;
- custom collection implementing **INotifyCollectionChanged** interface. If the collection keeps objects, implementing the **INotifyPropertyChanged** interface, two-way binding is supported;

- custom collection implementing the IList or IEnumerable interface.

Binding AppointmentStorage to a Collection of Custom Business Objects

1. Create a new Silverlight application and add an instance of C1Scheduler to the Page.
2. Add a new code file to your application and define a custom business object class. Note that the following definition only shows a public interface of this class; the full definition can be found in the C1Scheduler_BusinessObjectsBinding sample included with this product):

C#

```
// Business object should implement INotifyPropertyChanged interface.
public class AppointmentBORow : INotifyPropertyChanged
{
    public AppointmentBORow();
    public string Subject { get; set; }
    public DateTime Start { get; set; }
    public DateTime End { get; set; }
    public string Body { get; set; }
    public string Location { get; set; }
    public string Properties { get; set; }
    public Guid Id { get; set; }
    public bool IsDeleted { get; set; }
    public event PropertyChangedEventHandler PropertyChanged;
}
```

3. Add a class that will hold collection of **AppointmentBORow** objects:

C#

```
// Business objects collection should implement
// INotifyCollectionChanged interface.
public class AppointmentBOList : ObservableCollection<AppointmentBORow>
{
}
```

4. Add a custom data source class:

C#

```
public class AppointmentsBO
{
    AppointmentBOList _list = new AppointmentBOList();

    public AppointmentsBO(){ }

    public AppointmentBOList Appointments
    {
        get
        {
            return _list;
        }
```

```
        set
        {
            _list = value;
        }
    }
}
```

5. Create an instance of the **AppointmentsBO** class on a Page containing the C1Scheduler object:

XAML

```
<UserControl
    xmlns:c1sched="clr-
namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule"
    x:Class="C1Scheduler_BusinessObjectsBinding.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:local="clr-namespace:C1Scheduler_BusinessObjectsBinding"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
Loaded="UserControl_Loaded"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480" FontFamily="Segoe UI">
    <UserControl.Resources>
        <!-- Create instance of custom class keeping collection of business objects. -->
        <local:AppointmentsBO x:Key="_ds" />
</UserControl.Resources>
```

6. Use NestedPropertySetters to set mappings between the properties of **AppointmentBORow** objects and AppointmentStorage and to set the **AppointmentStorage.DataSource** property:

XAML

```
<c1sched:C1Scheduler x:Name="sched1" >
    <!-- Map AppointmentStorage to collection of business objects -->
    <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.AppointmentProperties.MappingName"
Value="Properties"/>
    <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Body.MappingName"
 Value="Body"/>
    <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.End.MappingName"
 Value="End"/>
    <c1sched:
PropertyName="DataStorage.AppointmentStorage.Mappings.IdMapping.MappingName"
Value="Id"/>
    <c1sched:
PropertyName="DataStorage.AppointmentStorage.Mappings.Location.MappingName"
Value="Location"/>
    <c1sched: PropertyName="DataStorage.AppointmentStorage.Mappings.Start.MappingName"
Value="Start"/>
    <c1sched: PropertyName="DataStorage.AppointmentStorage.Mappings.Subject.MappingName"
Value="Subject"/>
    <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataSource"
 Value="{Binding Path=Appointments, Mode=TwoWay, Source={StaticResource _ds}}" />
```

```
</c1sched:C1Scheduler>
```

That's all you need to do. At run time, any change to the **AppointmentBOList** will be reflected in C1Scheduler and vice versa.

Data-centric Architecture with Silverlight

Silverlight can be used to build line-of-business and other data-centric applications. This type of application typically involves the following steps:

1. Get the data from the server.
2. Display and edit the data on the client.
3. Submit the changes back to the server.

Steps 1 and 3 typically rely on Web services to transfer the data and traditional data access strategies to query and update a database. Step 2 typically involves Silverlight data-bound controls.

Microsoft offers many tools that can be used to perform the server-side part of the job. The latest such tool is ADO.NET Data Services, which provides Web-accessible endpoints for data models and integrates with Silverlight through the ADO.NET Data Services for Silverlight library (System.Data.Services.Client.dll). A lot has been written lately about this new technology (see for example "Data Services" in MSDN vol. 23, no. 10, September 2008).

ADO.NET Data Services is a powerful new technology that is likely to become a standard for many types of data-centric applications. However, it's not the only option. The traditional data ADO.NET classes (**DataSet**, **DataTable**, **DataAdapters**, and so on) can also be used to retrieve and update data on the server. These classes have been used by developers since .NET 1.0. They are solid, powerful, and easy to use. Furthermore, many developers already have a considerable investment in the form of code that has been used and tested for a long time.

Data for Silverlight is a set of classes that can be used by Silverlight clients to exchange data with servers using traditional ADO.NET. The typical steps are as follows:

1. **Get the data from the server.**
    a. Server populates a **DataSet** object the traditional way (typically using **DataAdapter** objects to retrieve the data from a SqlServer database).
    b. Server calls **DataSet.WriteXml** to serialize the **DataSet** into a stream and sends the stream to the client.
    c. Client uses the **DataSet.ReadXml** method to de-serialize the stream.
2. **Display and edit the data on the client.**
    a. Client binds the tables in the **DataSet** to controls (possibly using LINQ queries).
    b. User interacts with the controls to view, edit, add, and delete data items.
3. **Submit the changes back to the server.**
    a. Client calls **DataSet.GetChanges** and **DataSet.WriteXml** to serialize the changes into a stream and sends the stream to the server.
    b. Server calls **DataSet.ReadXml** to de-serialize the changes, then uses **DataAdapter** objects to persist the changes into the database.

The role of **C1Data** in this scenario is twofold. First, it provides a symmetric serialization mechanism that makes it easy to transfer data between **DataSet** objects on the client and on the server. Second, it provides a familiar object model to manipulate the data on the client.

> **Note: C1Data does not compete against ADO.NET Data Services**. It allows you to leverage your knowledge of ADO.NET and assets you may already have. You can transfer all that to Silverlight with minimal effort, and migrate to ADO.NET Data Services gradually, if such a migration is desired.
>
> **C1Data is not a legacy technology**. You can use it with LINQ, for example. In fact, **C1Data** enables LINQ features that are available on desktop but not in Silverlight applications (partial anonymous classes).

The next sections describe the implementation of a simple application that performs the steps described above. Despite its simplicity, the application shows how to perform the four CRUD operations (Create, Read, Update, Delete) that are required of most data-centric applications.

The Sample Application

The sample application will consist of a schedule. The data will be retrieved from the server when the application starts, using a Web service. It will then populate the schedule so the user can browse and edit the data. Finally, it will submit any changes back to the server so that the database can be updated.

Create the Application

Let's start by creating a new Silverlight application called "Scheduler". Complete the following steps:

1. Select **File | New | Project** to open the **New Project** dialog box in Visual Studio 2008.
2. In the **Project types** pane, expand either the **Visual Basic** or **Visual C#** node and select **Silverlight**.
3. Choose **Silverlight Application** in the **Templates** pane.
4. Name the project "Scheduler", specify a location for the project, and click **OK**. Next, Visual Studio will prompt you for the type of hosting you want to use for the new project.
5. In the **New Silverlight Application** dialog box, select **OK** to accept the default name ("Scheduler.Web") and settings and create the project.

Add the References

Our project will use two additional assemblies, **C1.Silverlight.Data.dll** (contains the **C1Data** classes) and **System.Windows.Controls.Data.dll** (contains Microsoft's **DataGrid** control). We chose to use the Microsoft grid instead of ComponentOne's to show that the **C1Data** classes are not tied in any way to other ComponentOne controls; they can be used against any Silverlight control.

To add the assemblies, complete the following steps:

1. Right-click the **Scheduler** project in the Solution Explorer and select **Add Reference**.
2. In the **Add Reference** dialog box locate and select the following assemblies and click **OK** to add references to your project:

   - C1.Silverlight.dll
   - C1.Silverlight.Data.dll
   - C1.Silverlight.DateTimeEditors.dll
   - C1.Silverlight.Schedule.dll

Create the UI

The user interface will contain a schedule.

To create the user interface, open the **MainPage.xaml** file and in Source view copy the following XAML onto the page:

XAML

```
<UserControl
    x:Class="BindingScheduler.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:c1sched="clr-
namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule"
    xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
    mc:Ignorable="d" d:DesignWidth="640" d:DesignHeight="480">
  <Grid x:Name="LayoutRoot">

<c1sched:C1Scheduler></c1sched:C1Scheduler>
 </Grid>
</UserControl>
```

Try running the application now to make sure everything is OK so far. You should see an empty schedule.

Implement the Server Side

In this step, we will implement the server side of the application. It consists of a Web service with two methods: **GetData** and **UpdateData**. **GetData** gets the data from the Schedule.mdb database and returns it to the client. **UpdateData** saves the changes made by the client back into the database. Note that the database can be found in C1Scheduler_DataBinding sample that is installed with this product.

Add the Database Access Infrastructure

Before implementing the Web services, let's add the database itself to the project. This step is optional. We will use a connection string to specify where the database is located, so we could use any existing copy of the MDB file already in the system. We will create a local copy here to facilitate deployment and to avoid making changes to the original database.

To add the database to the project, complete the following steps:

1. Right-click the **App_Data** node in the **Scheduler.Web** project and select **Add | Existing Item**.
2. In the dialog box, locate the **Schedule.MDB** file and add it to the project.
3. In the Properties window, set the **Build Action** property for the .**MDB** file to **None**.

In addition to the database file, we need a mechanism to transfer data from and to the database. In this sample, we will accomplish this using a utility class called **SmartDataSet**. **SmartDataSet** extends the regular ADO.NET **DataSet** class and adds the following:

- A **ConnectionString** property that specifies the database type and location.
- A **Load** method that loads data from selected tables.
- An **Update** method that saves changes back into the database.

The **SmartDataSet** is convenient but not necessary. It knows how to create and configure **DataAdapter** objects used to load and save data, but you could also write standard ADO.NET code to accomplish the same thing. Because it only

uses standard ADO.NET techniques, we will not list it here.

To add the **SmartDataSet.cs** file to the project, complete the following:

1. Right-click the **Scheduler.Web** node in the Solution Explorer and select **Add | Existing Item**.
2. In the **Add Existing Item** dialog box, locate the **SmartDataSet.cs** file in the **C1.Silverlight** distribution package and click **Add** to add it to the project.

Implement the Web Service

The server side of the application consists of Web services that load the data from the database and returns it to the server, or receive a list of changes from the client and applies it to the database.

We could implement the server side as a generic handler class (ashx), as a Web Service (asmx), or as a Silverlight-enabled WCF service (SVC). For this sample, any of the choices would do. We will choose a classic Web Service.

Follow these steps to create the service:

1. Right-click the **Scheduler.Web** project.
2. Select **Add | New Item**.
3. In the **Add New Item** dialog box, select **Web** from the list of **Categories** in the left pane.
4. In the right pane, select the **Web Service** template from the list of templates.
5. Name the new service "DataService.asmx".
6. Click the **Add** button to create the new Web Service.

After you click the **Add** button, Visual Studio will open the newly created **DataService.asmx.cs** (or **DataService.asmx.vb**) file. The file contains a single **HelloWorld** method.

Edit the file as follows:

1. Add the following import statements to the statements block at the start of the file:

C#

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

2. Uncomment the line below so the service can be called from Silverlight:

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

3. Delete the **HelloWorld** method that Visual Studio created and replace it with the following code:

C#

```
[WebMethod]
public byte[] GetData(string tables)
{
  // Create DataSet with connection string
  var ds = GetDataSet();

  // Load data into DataSet
  ds.Fill(tables.Split(','));

  // Persist to stream
  var ms = new System.IO.MemoryStream();
```

```
  ds.WriteXml(ms, XmlWriteMode.WriteSchema);

  // Return stream data
  return ms.ToArray();
}
```

The method starts by creating a **SmartDataSet**, then fills it with the tables specified by the **tables** parameter. Finally, it uses the **WriteXml** method to persist the **DataSet** into a stream, converts the stream into a byte array, and returns the result.

Remember that this code will run on the server. You could use a data compression library such as **C1.Zip** to compress the stream and reduce its size significantly before returning it to the client. We did not bother to do this here to keep the example as simple as possible.

4. Next, add the method that saves the changes back into the database with the following code:

C#

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

The method takes three parameters, each corresponding to a different type of change to be applied to the database: records that were added, modified, and deleted. It calls the **UpdateData** helper method to apply each set of changes, and returns null if all changes were applied successfully. If there are any errors, the method returns the a message that describes the exception.

5. Add the following code for the **UpdateData** helper method:

C#

```
void UpdateData(byte[] data, DataRowState state)
{
  // No changes, no work
  if (data == null)
    return;

  // Load data into dataset
  var ds = GetDataSet();
  var ms = new MemoryStream(data);
  ds.ReadXml(ms);
```

```
  ds.AcceptChanges();

  // Update row states with changes
  foreach (DataTable dt in ds.Tables)
  {
    foreach (DataRow dr in dt.Rows)
    {
      switch (state)
      {
        case DataRowState.Added:
          dr.SetAdded();
          break;
        case DataRowState.Modified:
          dr.SetModified();
          break;
        case DataRowState.Deleted:
          dr.Delete();
          break;
      }
    }
  }

  // Update the database
  ds.Update();
}
```

The method starts by creating a **SmartDataSet** and loading all the changes into it. It then changes the **RowState** property on each row to identify the type of change that has been applied to the row (added, modified, or deleted). Finally, it calls the **SmartDataSet.Update** method to write the changes to the database.

6. The server-side code is almost ready. The only part still missing is the method that creates and configures the **SmartDataSet**. Add the following code for the implementation:

C#

```
Type yoSmartDataSet GetDataSet()
{
  // Get physical location of the mdb file
  string mdb = Path.Combine(
    Context.Request.PhysicalApplicationPath, @"App_Data\Schedule.mdb");

  // Check that the file exists
  if (!File.Exists(mdb))
  {
    string msg = string.Format("Cannot find database file {0}.", mdb);
    throw new FileNotFoundException(msg);
  }

  // Make sure file is not read-only (source control often does this...)
  FileAttributes att = File.GetAttributes(mdb);
  if ((att & FileAttributes.ReadOnly) != 0)
```

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

  // Create and initialize the SmartDataSet
  var dataSet = new SmartDataSet();
  dataSet.ConnectionString =
    "provider=microsoft.jet.oledb.4.0;data source=" + mdb;
  return dataSet;
}
```

The method starts by locating the database file, making sure it exists, and checking that it is not read-only (or the updates would fail). Once that is done, it creates a new **SmartDataSet**, initializes its **ConnectionString** property, and returns the newly created **SmartDataSet** to the caller.

Implement the Client Side

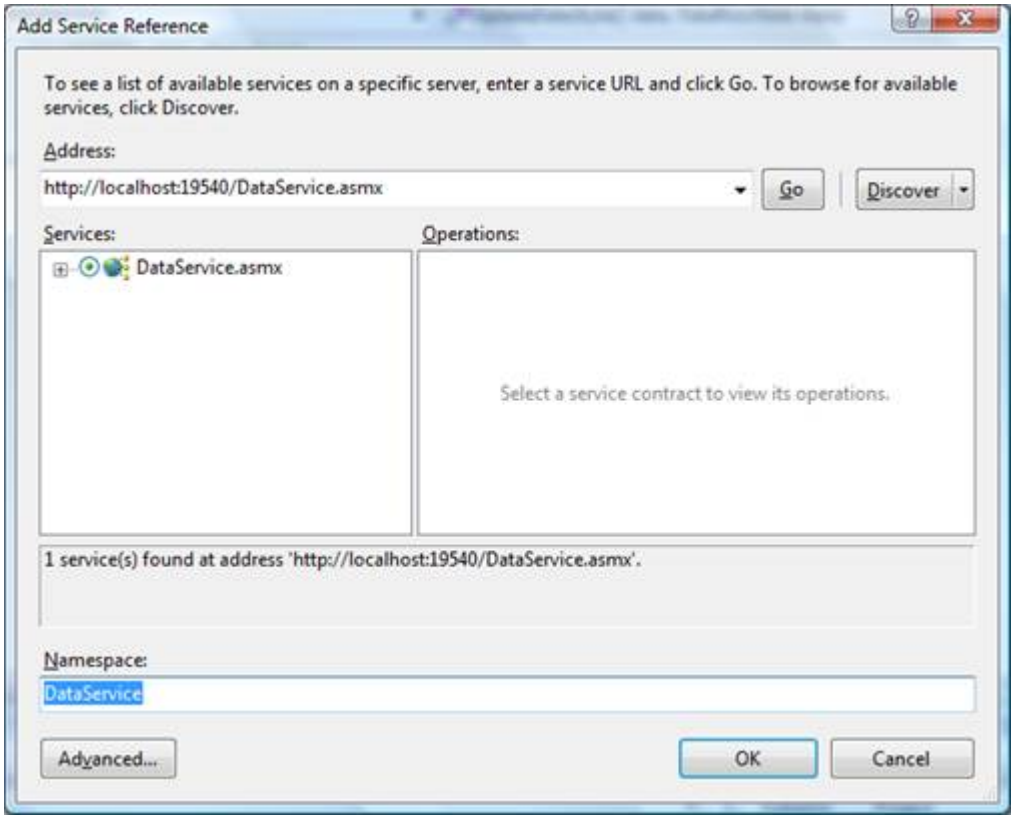
Now let's return to the client side of the application. In the following sections you'll implement the client side.

Add a Reference to the Web Service

The first thing we will do is add a reference to the Web service we just created, so we can invoke it from the client. Complete the following steps:

1. Right-click the **References** node in the **Scheduler** project and select **Add Service Reference**.
2. In the dialog box that follows, click the **Discover** button. Visual Studio will locate the service we just added in the previous step.
3. In the **Namespace** field, type "DataService".

   The dialog box should look like this:



4. Click **OK** to add the reference.

Use the Web Service to Get the Data

We are finally ready to get the data and show it to the users. Complete the following:

1. Open the **MainPage.xaml.cs** file and add the following using statements to the block at the top of the file:

```
C#
using System.IO;
using C1.Silverlight.Data;
using Scheduler.DataService;
using C1.Silverlight.Schedule;
using C1.C1Schedule;
```

2. Next, edit the page constructor as follows:

```
C#
private DataSet _ds = null;
private DataTable _dtAppointments = null;

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

```
  // set mappings for AppointmentStorage
  // (set mapping between AppointmentStorage and Appointment table columns)
  AppointmentMappingCollection mappings =
sched1.DataStorage.AppointmentStorage.Mappings;
  mappings.IdMapping.MappingName = "Id";
  mappings.Subject.MappingName = "Subject";
  mappings.Body.MappingName = "Body";
  mappings.End.MappingName = "End";
  mappings.Start.MappingName = "Start";
  mappings.Location.MappingName = "Location";
  mappings.AppointmentProperties.MappingName = "Properties";

  // load actual data from server
  LoadData();
}
```

3. Add the following code to implement the **LoadData** method to invoke the Web service to retrieve the data into a **DataSet** object, which we will then bind to the controls on the page:

C#

```
DataSet _ds = null;
void LoadData()
{
  // Invoke Web service
  var svc = new GetDataService();
  svc.GetDataCompleted += svc_GetDataCompleted;
  svc.GetDataAsync("Appointments");
}
void svc_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
{
  // Handle errors
  if (e.Error != null)
  {
     tbStatus.Text = "Error downloading data...";
     return;
  }

  // Parse data stream from server (DataSet as XML)
  tbStatus.Text = string.Format(
  "Got data, {0:n0} kBytes", e.Result.Length / 1024);

  var ms = new MemoryStream(e.Result);
  ds = new DataSet();
  ds.ReadXml(ms);

  // Got the data, find the table
  dtAppointments = _ds.Tables["Appointments"];

  // set C1Scheduler data source to the DataTable loaded from server
  sched1.DataStorage.AppointmentStorage.DataSource = _dtAppointments;
}
```

4. Add the following **GetDataService** method implementation:

C#

```
// Get data service relative to current host/domain
DataServiceSoapClient GetDataService()
{
  // Increase buffer size
  var binding = new System.ServiceModel.BasicHttpBinding();
  binding.MaxReceivedMessageSize = 2147483647; // int.MaxValue
  binding.MaxBufferSize = 2147483647; // int.MaxValue

  // Get absolute service address
  Uri uri = GetAbsoluteUri("DataService.asmx");
  var address = new System.ServiceModel.EndpointAddress(uri);

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

The **GetDataService** method instantiates and returns a new **DataServiceSoapClient** object. We don't use the default constructor because that would refer to the development environment (http://localhost and so on). It would work correctly on the development machine, but would break when the application is deployed. Also, the default constructor uses a 65k buffer that might be too small for our data transfers. The above **GetDataService** method implementation takes care of both issues.

The **LoadData** method above instantiates the service and invokes the **GetDataAsync** method. When the method finishes executing, it invokes the **svc_DataCompleted** delegate. The delegate instantiates a **DataSet** object, uses the **ReadXml** method to de-serialize the data provided by the server, and then sets datasource for AppointmentStorage. As mappings for AppointmentStorage have been set in a Page constructor, setting the **AppointmentStorage.DataSource** property will bind data to the C1Scheduler control. That would be a two-way binding, so any change to _dtAppointments will be reflected in C1Scheduler and vice versa.

Note: This is one of the most important features of the **C1.Silverlight.Data DataSet** class. It uses the same XML schema that ADO.NET **DataSet** objects use. This allows applications to serialize data on the client and de-serialize it on the server, or vice-versa. The fact that the object models are very similar also makes things substantially easier for the developer.

Commit Changes to the Server

We are almost done now. The only piece still missing is the code that submits the changes made by the user back to the server, so they can be applied to the database.

The first step is to add a button for commiting changes to server. The event handler for this button will save data back to server:

C#

```
// Commit changes to server
private void _btnCommit_Click(object sender, RoutedEventArgs e)
{
  SaveData();
}
```

And here is the implementation for the **SaveData** method, which does the real work:

C#

```
// Save data back into the database
void SaveData()
{
  if (_ds != null)
  {
    // get changes of each type
    byte[] dtAdded = GetChanges(DataRowState.Added);
    byte[] dtModified = GetChanges(DataRowState.Modified);
    byte[] dtDeleted = GetChanges(DataRowState.Deleted);

    // Invoke service
    var svc = new GetDataService();
    svc.UpdateDataCompleted += svc_UpdateDataCompleted;
    svc.UpdateDataAsync(dtAdded, dtModified, dtDeleted);
  }
}
void svc_UpdateDataCompleted(object sender, UpdateDataCompletedEventArgs e)
{
  if (!string.IsNullOrEmpty(e.Result))
  {
    throw new Exception("Error updating data on the server: " + e.Result);
  }
  _tbStatus.Text = "Changes accepted by server.";
  _ds.AcceptChanges();
}
```

The method starts by calling the **GetChanges** method to build three byte arrays. Each one represents a **DataSet** with the rows that have been added, modified, or deleted since the data was downloaded from the server. Then the method invokes the Web service we implemented earlier and listens for the result. If any errors were detected while updating the server, we throw an exception (real applications would deal with the error in a more elegant way).

The only piece still missing is the **GetChanges** method. Here it is:

C#

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

```
            return ms.ToArray();
        }
        return null;
    }
```

The method uses the **DataSet.GetChanges** method to obtain a new **DataSet** object containing only the rows that have the **DataRowState** specified by the caller. This is the same method available on the ADO.NET **DataSet** class.

The method then serializes the **DataSet** containing the changes into a **MemoryStream**, and returns the stream contents as a byte array.

Try running the application now. Make some changes, then click the "Commit Changes" button to send the changes to the server. If you stop the application and start it again, you should see that the changes were indeed persisted to the database.

Adding Localized Resources to a Project

Scheduler for WPF and Silverlight projects can be localized. To add localized resources to your project:

1. Create a copy of the .resx files included with Scheduler for WPF and Silverlight. These resource files are installed in **C:\Program Files\ComponentOne\WPF Edition\C1WPFNewSchedule\DefaultResources** and **C:\Program Files\ComponentOne\Silverlight Edition\Samples\C1.Silverlight.Schedule\C1Scheduler_Localization\DefaultResources** by default.
2. Rename the copied resource files using the following file naming convention:
   `base_filename[.optional RFC 1766 culture info string].resx`

   For example:

   TimeStrings.de.resx

   TimeStrings.fr.resx

   > **Note:** Do not change the base_filename part of the file name or the controls will not be able to find your resources.

3. Translate the string values within the resource files.
4. Add the localized resources to your project /Resources folder. Open the .csproj file in the text editor and add your culture to the list of supported cultures:

   <SupportedCultures>de;fr </SupportedCultures>.

After rebuilding your project, satellite assemblies will be added into the resulting .xap file.

The following resource files are used in the C1.Silverlight.Schedule assembly:

Resource File	Description
C1.Schedule.Categories.resx	Contains localizable names for default appointment categories.
C1.Schedule.EditAppointment.resx	Contains localizable strings for default EditAppointmentTemplate.
C1.Schedule.EditRecurrence.resx	Contains localizable strings for default EditRecurrenceTemplate.
C1.Schedule.Exceptions.resx	Contains localizable error messages which might be shown to end user.
C1.Schedule.Labels.resx	Contains localizable names for default appointment labels.
C1.Schedule.MiscStrings.resx	Contains localizable strings used for describing recurrence pattern.

C1.Schedule.RecChoice.resx	Contains localizable strings for Open/Remove recurrence dialogs.
C1.Schedule.SelectFromListScene.resx	Contains localizable strings for SelectFromListScene control.
C1.Schedule.ShowReminders.resx	Contains localizable strings for default ShowRemindersTemplate.
C1.Schedule.Statuses.resx	Contains localizable names for default availability statuses.
C1.Schedule.TimeStrings.resx	Contains localizable strings for the C1TimeSpanPicker control.
C1.Silverlight.resx	Contains localizable strings for the C1MessageBox control.

Note: You can localize only the part of these files. For example, if you use customEditAppointmentTemplate, you do not need resources from the C1.Schedule.EditAppointment.resx.

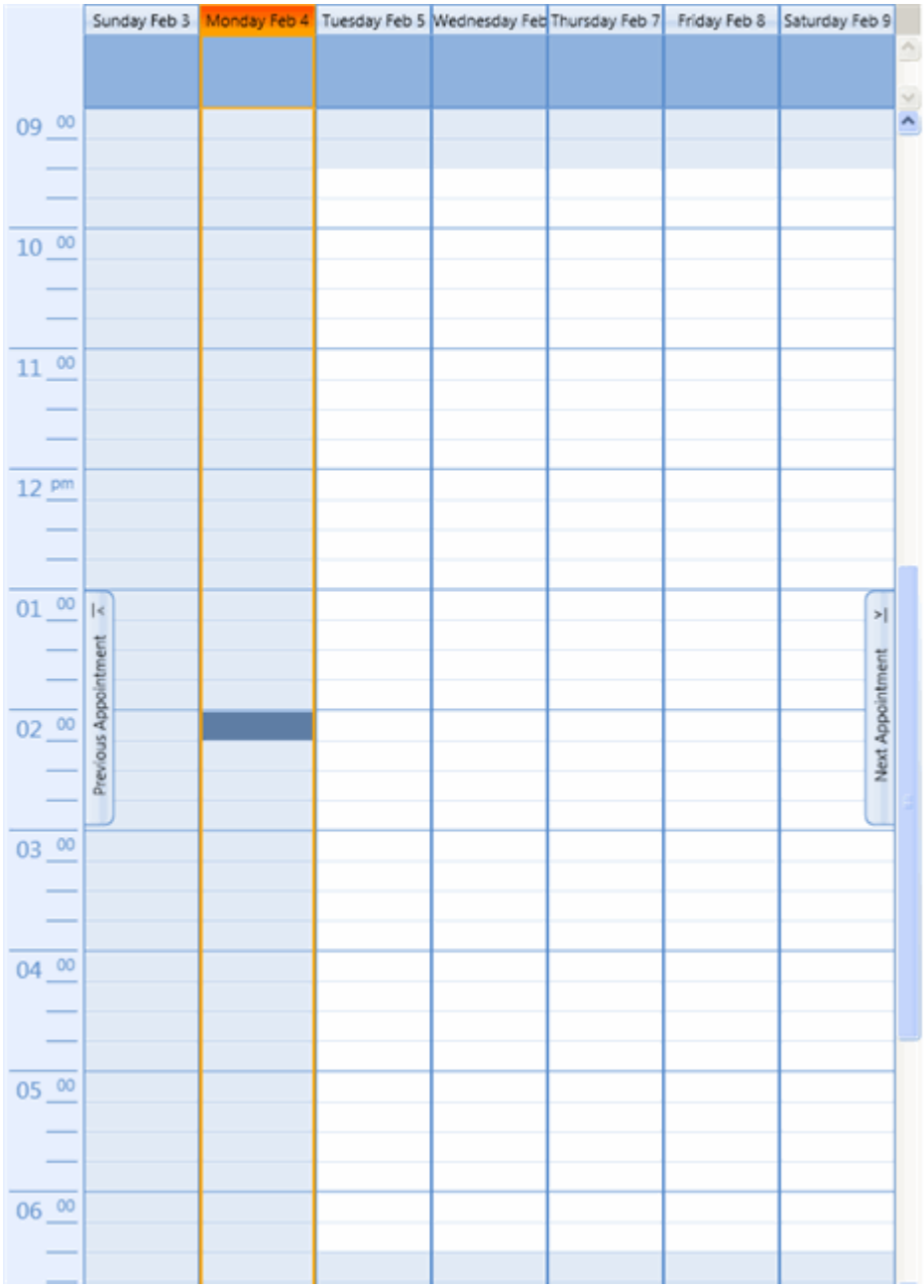
Calendar Settings

The **C1Scheduler** control uses the set of calendar settings defined via the corresponding properties of the CalendarHelper class. For example, you can create a work week calendar by specifying the work days through the CalendarHelper class. Notice that the XAML for the CalendarHelper class is the same for all three controls.

XAML for C1Scheduler

XAML

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



The following calendar settings are available:

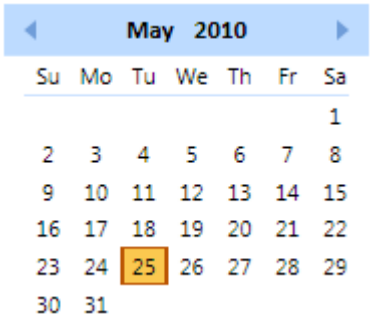
CalendarHelper Property	Description
WeekStart	Gets or sets the DayOfWeek value determining the first day of the week. The default is system settings.
WorkDays	Gets or sets the WorkDays object containing the set of working days in one week.
StartDayTime	Gets or sets the TimeSpan value specifying the beginning of the working time.
EndDayTime	Gets or sets the TimeSpan value specifying the end of the working time.
Holidays	Gets or sets ObservableCollection<DateTime> object which holds the list of holidays (non-working days in addition to weekends).
WeekendExceptions	Gets or sets the ObservableCollection<DateTime> object which holds the list of

	weekend days which should be working.
FullMonthNames	Gets an array of culture specific full month names.

Using the C1Calendar Control

C1Scheduler provides one supplementary calendar control: C1Calendar.

The C1Calendar control is used to create a one-month or multiple month calendar user interface. It enables users to select a specific date(s) interactively.



The purpose of the calendar control is to provide data that helps to create the calendar user interface (without code, solely in XAML) with an ability to select a specific date interactively. The main properties that express this purpose are:

Main Calendar Properties

Property	Description
CalendarBase.SelectedDate DateTime SelectedDate {get; set;}	Defines a current date that is selected in calendar.
CalendarBase.Year int Year {get; set;}	Defines a year component of the C1.WPF.Schedule.CalendarBase.SelectedDate property.
CalendarBase.Month int Month {get; set;}	Defines a month that is represented by a calendar.
CalendarBase.MaxDate DateTime MaxDate {get; set;}	Gets or sets the maximum allowable date. The default value is 12/31/9998.
CalendarBase.MinDate TimeSpan MinDate {get; set;}	Gets or sets the minimum allowable date. The default value is 01/01/1753.

These properties are being kept in synch, so changing the DateTime changes the Year and Month appropriately and vice versa.

Other Calendar Properties

Property	Description
CalendarBase.CalendarHelper, public C1.WPF.Schedule.CalendarHelper CalendarHelper {get; set;}	Provides calendar-dependant properties.
C1Calendar.MaxSelectionCount, public int MaxSelectionCount {get; set;}	Defines the maximum number of days that can be selected in the control.

C1Calendar.SelectedDates, public C1.WPF.Schedule.DateList SelectedDates {get; set;}	The list of selected dates.
C1Calendar.BoldedDates, public C1.WPF.Schedule.DateList BoldedDates {get; set;}	The list of bolded dates.
CalendarBase.DaysPanel, public System.Windows.Controls.ItemsPanelTemplate DaysPanel {get; set;}	An ItemsPanelTemplate that defines the panel that lays out elements representing days of a month. By default the AutoDistributionGrid panel with 7 columns and 6 rows is used.
CalendarBase.DaySlotTemplate, public System.Windows.DataTemplate DaySlotTemplate {get; set;}	A DataTemplate that defines a UI representation of a single day of a month. A DataContext for this template is a DaySlot object.
CalendarBase.DaySlotStyle, public System.Windows.Style DaySlotStyle {get; set;}	A Style for DaySlotPresenter elements which are the root elements of a visual tree representing a single day of a month.
CalendarBase.DaysOfWeekPanel, public System.Windows.Controls.ItemsPanelTemplate DaysOfWeekPanel {get; set;}	An ItemsPanelTemplate that defines the panel that lays out elements representing days of week. By default the StackPanel with horizontal orientation is used.
CalendarBase.DayOfWeekSlotTemplate, public System.Windows.DataTemplate DayOfWeekSlotTemplate {get; set;}	A DataTemplate that defines a UI representation of a single day of week. A DataContext for this template is a DayOfWeekSlot object.
C1CalendarItem.MonthFullName, public string MonthFullName { get; }	Gets a full name of a month currently represented by calendar taking into account current culture.
CalendarBase.Theme, public System.Windows.ResourceDictionary Theme {get; set;}	Gets or sets a ResourceDictionary object containing calendar theme resources.

All controls expose the following RoutedEvent:

Event	**Description**
CalendarBase.SelectedDateChanged	Occurs when the **C1.WPF.Schedule.CalendarBase.SelectedDate** property value has been changed.

C1Calendar Elements

User interface abstraction of C1Calendar implies an existence of three main parts:

1. A pane that represents a list of the month's days
2. A pane that represents a list of days of the week names
3. A command pane that is intended to represent a UI that manages a selection of the current month/year

A list of days is represented by the C1CalendarItemPresenter object, which is inherited from the **ListBox** class. An instance of C1CalendarItemPresenter class is used in the C1Calendar's template visual tree to define a place where a

panel with calendar days will appear.

C1Calendar generates UI elements (of the DaySlotPresenter class) representing calendar day cells for the CalendarBase.Year and CalendarBase.Month properties. An actual UI of these DaySlotPresenter elements is defined in the C1CalendarResources.DaySlotTemplate property. Those elements become children of a panel whose UI is defined in the CalendarBase.DaysPanel property.

Note: A regular calendar contains 6 week rows; each row contains 7 days; therefore, 6 * 7 = 42 day cells (referred to as "slots"). Some cells don't represent any day - they are just empty cells.

Calendar Day Cells (Slots)

Each DaySlotPresenter element gets a DaySlot type object as its DataContext. This allows convenient binding of the DaySlotPresenter UI to a DaySlot object. DaySlot provides information about a certain day cell, which includes the following read-only properties:

- bool DaySlotPresenter.Empty - indicates whether a slot represents a day or it's an empty one
- DateTime DaySlotPresenter.Date - gets a date represented by the DaySlot or a null value if the DaySlot is empty
- DayOfWeekDaySlot.DayOfWeek - gets the day of the week to which this slot corresponds
- bool DaySlot.IsWeekEnd - indicates whether this day is a weekend
- bool DaySlot.IsSelected - indicates whether this day slot is currently selected
- bool DaySlot.IsAdjacent - indicates whether the DaySlot represents a day from an adjacent month, but not of the month currently represented by the C1Calendar
- bool DaySlot.IsBolded - indicates whether a day represented by the DaySlot is currently bolded in the C1Calendar UI

Day Slot Generation

Day slots are generated with regard to culture-specific day of the week ordering. For example, if the first day of the month is a Tuesday, then the corresponding day slot will be third in the list, with two empty slots preceding it, for the USA culture, because Sunday is the first day of the week in the USA; however, it will be second in the list for the Russian culture, because Monday is the first day of the week in Russia. A represented culture is controlled by the **C1Calendar** CalendarBase.CalendarHelper property.

To specify a place in the calendar control UI tree (defined in the **C1Calendar.Template**) where the abovementioned panel with days will be placed, the C1CalendarPresenter type instance should be used.

Days of Week

The days of the week list is represented by the DaysOfWeekPresenter class derived from ItemsControl. An instance of DaysOfWeekPresenter should be used as a placeholder in the **C1Calendar.Template** visual tree to specify where the days of the week pane should be placed. DaysOfWeekPresenter generates 7 DaysOfWeekPresenter objects as children of a panel whose template is defined in the C1Calendar CalendarBase.DaysOfWeekPanel.
Each DaysOfWeekPresenter object represents a single day of the week. A UI of each DayOfWeekSlotPresenter is defined in the C1Calendar CalendarBase.DayOfWeekSlotTemplate property. Each DayOfWeekSlotPresenter receives a DayOfWeekSlot object as its DataContext. DayOfWeekSlot provides a set of properties for convenient binding of

the DayOfWeekSlotPresenter UI, like the following:

- DayOfWeek DayOfWeekSlot.DayOfWeek – gets the day of the week represented by the DayOfWeekSlot
- string DayOfWeekSlot.DayFullName – the culture-specific full name of a day;
- string DayOfWeekSlot.DayShortName – the culture-specific abbreviated name of a day;
- string DayOfWeekSlot.DayShortestName – the culture specific shortest name for the DayOfWeek;
- bool DayOfWeekSlot.IsWeekEnd – indicates whether this day of the week is a weekend day.

The order of the represented days of the week in the list is culture-specific. For example, the first day of the week is Sunday in the USA culture and Monday in the Russian culture.

C1Calendar Appearance

There are several ways you can customize C1Calendar's appearance. The following topics discuss some these customization techniques, including brush properties, themes, and templates.

C1Calendar Properties

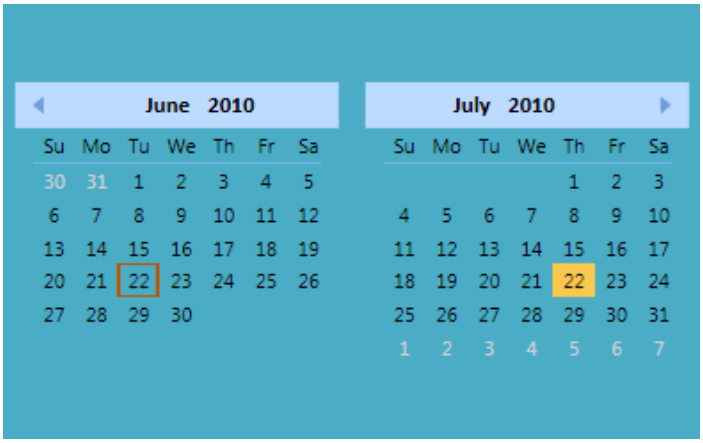
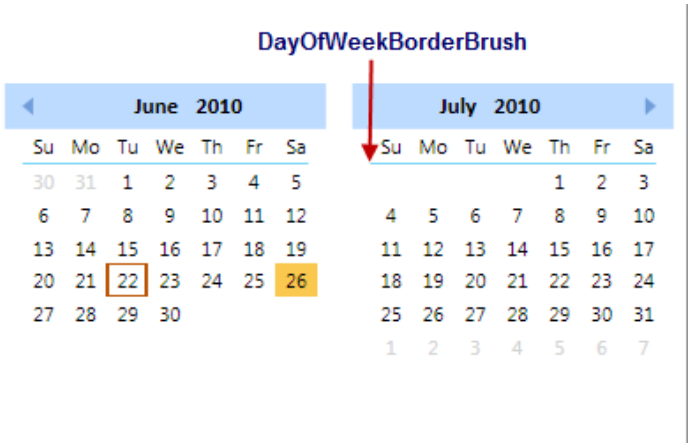
Scheduler for WPF and Silverlight includes several properties that allow you to customize the appearance of the C1Calendar control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties:

Color Properties

The following properties let you customize the colors used in the parts of the month area and month header of the calendar control such as the adjacent month days, days of week, month header, selected days, weekends, and the current day. Additionally it includes color properties for the foreground and background of the control itself.

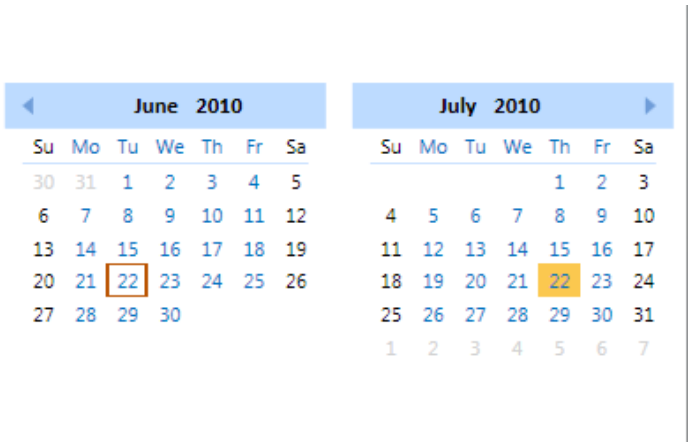
Property	Description
C1Calendar.AdjacentMonthDayBrush	Gets or sets a Brush object used to display adjacent month days. This is a dependency property.
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
C1Calendar.DaysOfWeekBorderBrush	Gets or sets a Brush object used to underline days of week. This is a dependency property.
Foreground	Gets or sets a brush that describes the foreground color. This is a dependency property.
C1Calendar.MonthHeaderBackground	Gets or sets a Brush object used to color month header. This is a dependency property.
C1Calendar.MonthHeaderForeground	Gets or sets a Brush object used to color month header text. This is a dependency property.

C1Calendar.NavigationButtonBrush	Get or sets a Brush used to color navigation buttons. This is a dependency property.
C1Calendar.SelectedDayBrush	Gets or sets a Brush object used to highlight selected dates. This is a dependency property.
C1Calendar.TodayBrush	Gets or sets a Brush object used to highlight current date. This is a dependency property.
C1Calendar.WeekendBrush	Gets or sets a Brush object used to display weekends. This is a dependency property.

Property	**Example**
C1Calendar.AdjacentMonthDayBrush	
Background	
C1Calendar.DaysOfWeekBorderBrush	

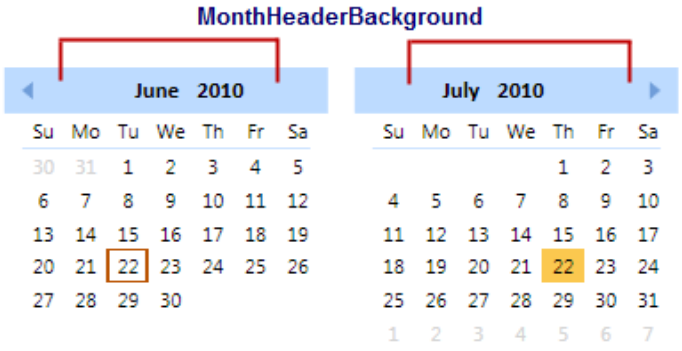
Foreground

This property effects the abbreviated work days (Monday through Friday) and the calendar number work days (Monday through Friday).



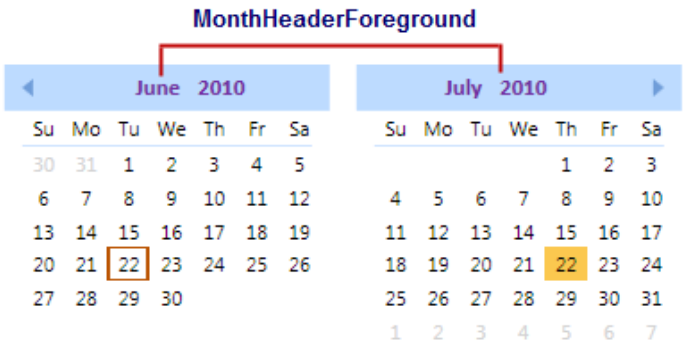
C1Calendar.MonthHeaderBackground

This property affects the background color for the month header.

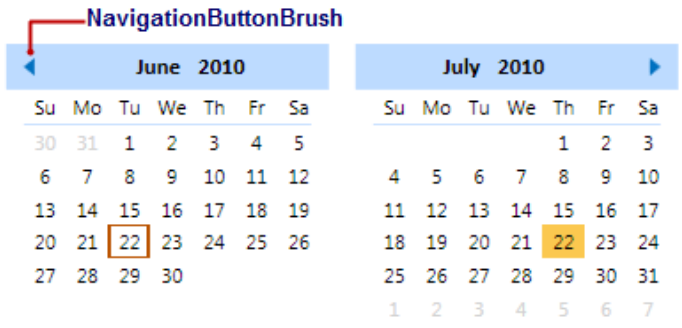


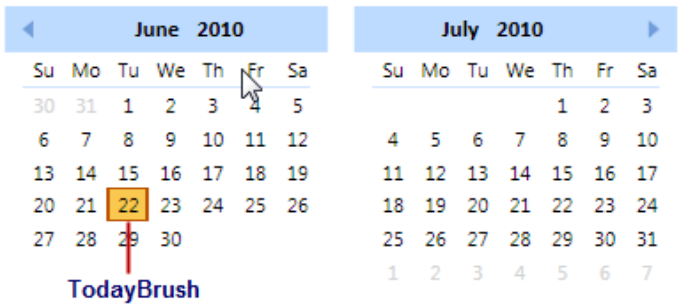
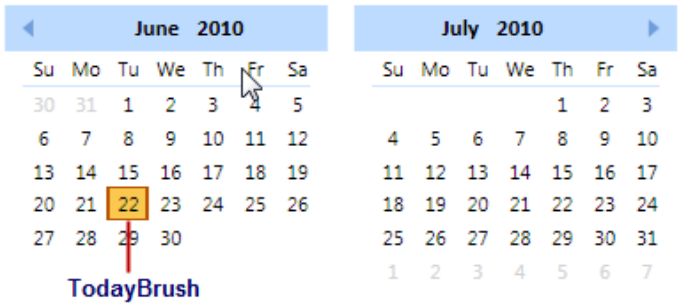
C1Calendar.MonthHeaderForeground

This property affects the Month name and Year that appears in the month header.



C1Calendar.NavigationButtonBrush



C1Calendar.SelectedDayBrush	
C1Calendar.TodayBrush	
C1Calendar.WeekendBrush	

Text Properties

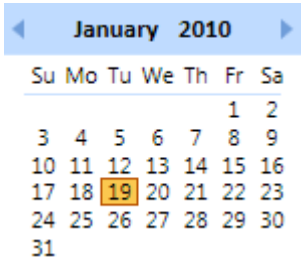
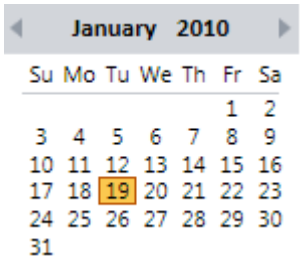
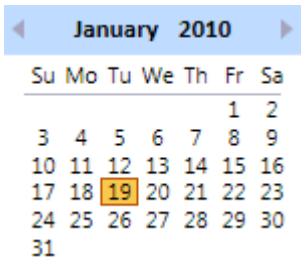
The following properties let you customize the appearance of text in the C1Calendar control.

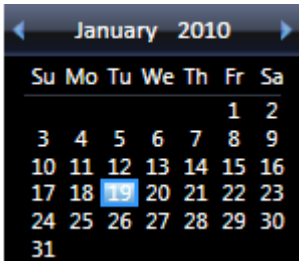
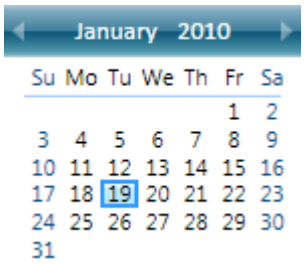
Property	Description
FontFamily	Gets or sets the font family of the control. This is a dependency property.

FontSize	Gets or sets the font size. This is a dependency property.
FontStretch	Gets or sets the degree to which a font is condensed or expanded on the screen. This is a dependency property.
FontStyle	Gets or sets the font style. This is a dependency property.
FontWeight	Gets or sets the weight or thickness of the specified font. This is a dependency property.
C1Calendar.MonthHeaderFontFamily	Gets or sets a FontFamily object used to display month header text. This is a dependency property.
C1Calendar.MonthHeaderFontSize	Gets or sets a Double value determining month header font size. This is a dependency property.
C1Calendar.MonthHeaderFontWeight	Gets or sets a FontWeight object used to display month header text. This is a dependency property.

C1Calendar Themes

The calendar control has seven predefined themes.

C1Scheduler Themes	Example
Office 2007 Blue	
Office 2007 Black	
Office 2007 Silver	

Media Player	
Dusk Blue	
Dusk Green	
Vista	

The most common user interface properties of calendar controls are defined in theme ResourceDictionaries. The following table lists default calendar themes:

Theme Folder	Theme File	Static Field of C1CalendarResources Class	ResourceID	Description
Office2007	Black.xaml	Office2007Black	Office2007.Black	Office 2007 Black theme.
	Blue.xaml	Office2007Blue	Office2007.Blue	Office 2007 Blue theme.
	Silver.xaml	Office2007Silver	Office2007.Silver	Office 2007 Silver theme.
Dusk	Blue.xaml	DuskBlue	Dusk.Blue	Dusk Blue theme.
	Green.xaml	DuskGreen	Dusk.Green	Dusk Green theme.
Media Player	MediaPlayer.xaml	MediaPlayer	MediaPlayer	Media Player theme.
Vista	Vista.xaml	Vista	Vista	Vista theme.

By default, the calendar themes are installed in the **Theme Folder** listed above within the **C:\Program Files\ComponentOne\WPF Edition\C1Schedule\XAML\themes\CalendarThemes** folder.

Setting the Calendar Theme

All calendar controls use the current system theme, by default. If you want to use a different theme, there are several ways to select a new one.

To set the theme at design time in Visual Studio:

1. Right-click the control.
2. Select **Theme** and choose one of the seven predefined themes.

> **Note:**You can also change the theme through the Properties window by selecting the option from the drop-down list next to the CalendarBase.Theme property.

To set the theme in Microsoft Blend, change the CalendarBase.Theme **property at design time:**

1. Select the C1Calendar control in your XAML window or page.
2. In the Properties panel, under **Appearance**, click the drop-down arrow next to the CalendarBase.Theme property and choose one of the predefined themes.

To set the theme using the ResourceID, use the following XAML:

XAML

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

To set the theme using C1CalendarResources static fields, add the following code to your project:

Visual Basic

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

C#

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

To set the theme by defining a ResourceDictionary and DefaultThemeKey in your Page, Window, or Application resources:

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>
```

Note that this will affect all controls in the current scope.

You can also create your own theme ResourceDictionaries and use them with the calendar controls.

The best way to customize one of the predefined themes is to include the default theme definition in a custom ResourceDictionary and redefine necessary resources, such as theme brushes, there.

Note: To ensure all default styles and templates continue to work correctly during customization, it is suggested that the resource keys are not changed from their default settings.

Default Calendar Theme Resources

The following table depicts the details of default calendar theme resources along with their keys:

Resource key	Description
C1Calendar_AdjacentDateText_Brush	The brush which is used for displaying adjacent days text.
C1Calendar_Background	The brush which is used as the calendar control's background.
C1Calendar_DaysOfWeekBorder_Brush	The brush which is used for coloring the line under weekday names.
C1Calendar_Font	The FontFamily which is used in the calendar control.
C1Calendar_FontSize	The double value determining the font size used by the control.
C1Calendar_MonthHeader_Font	The FontFamily which is used for displaying mant header.
C1Calendar_MonthHeaderFont_Size	The double value determining the font size used for displaying mant header.
C1Calendar_MonthHeader_Brush	The brush which is used for coloring month header background.
C1Calendar_MonthHeaderText_Brush	The brush which is used for coloring month header background.
C1Calendar_NavArrow_Brush	The brush which is used for coloring navigation arrows.
C1Calendar_SelectedDate_Brush	The brush which is used for coloring selected days background.
C1Calendar_TodayBorder_Brush	The brush which is used for coloring current date border.
C1Calendar_TodayBorder_Thickness	The border thickness of the current date border.
C1Calendar_WeekendText_Brush	The brush which is used for coloring weekend days.

Calendar Templates

To provide a custom look for the C1Calendar control:

1. To define a general layout model for a calendar, the **C1Calendar.Template** property should be assigned; this is usually done through a Setter of a Style. The template may contain any UI elements, for example, grids with StackPanels and borders with anything, but in some area of the template tree, the following placeholder elements of the calendar should be placed:
    - C1CalendarPresenter - to designate a place where a days pane will appear;
    - DaysOfWeekPresenter - to designate a place where a days of week pane will appear;

    Note that each of the placeholders enumerated above is optional.

2. To define the Days pane UI, assign templates to the following properties:
    - C1Calendar CalendarBase.DaysPanel – defines a panel for the layout of day items;
    - C1Calendar C1CalendarResources.DaySlotTemplate – defines a UI that represents each day. Bind the UI of this template using the "{Binding Path=DaySlot_Property}" markup extension, where DaySlot_Property is any property name of the DaySlot class;
    - C1Calendar C1CalendarResources.DaySlotStyle – allows you to define the properties of a root object of the UI tree representing a separate day. This root object is of the DaySlotPresenter type.
3. To define the Days of Week pane UI, assign templates to the following properties:
    - C1Calendar CalendarBase.DaysOfWeekPanel – defines a panel for the layout of days of week items;
    - C1Calendar CalendarBase.DayOfWeekSlotTemplate – defines a UI that represents each day of the week. Bind the UI of this template using the "{Binding Path=DayOfWeekSlot_Property}" markup extension, where DayOfWeekSlot Property is any property of the DayOfWeekSlot class;
    - C1Calendar CalendarBase.DayOfWeekSlotStyle – allows you to define the properties of a root object of the UI tree representing a separate day. This object is of the DayOfWeekSlotPresenter type.

C1Calendar Layout

There are several ways you can customize C1Calendar's layout. The following topics discuss some these customization techniques, including layout properties, multiple calendar presentation, and calendar layout.

C1Calendar Layout Properties

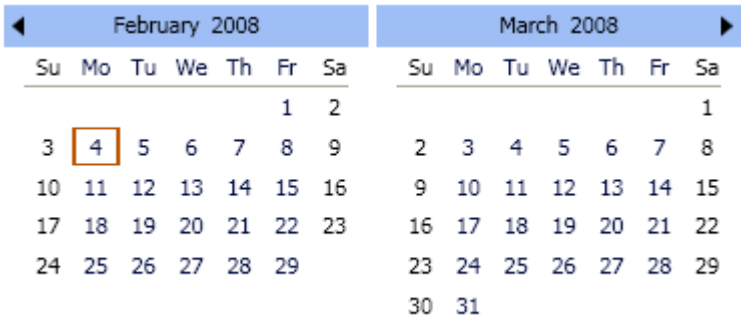
Calendar for WPF includes several properties that allow you to customize the layout of the calendar control. You can change the width, height, alignment, and the amount of calendar months displayed of the control. The following properties let you customize the control's layout:

Property	Description
Width	Gets or sets the Width of the element.
Height	Gets or sets the Height of the element.
HorizontalAlignment	Gets or sets the horizontal alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control.
VerticalAlignment	Gets or sets the vertical alignment characteristics applied to this element when it is composed within a parent element, such as a panel or items control.
Margin	Gets or sets the outer margin on an element.

Padding	Gets or sets the padding inside a control.
MinWidth	Gets or sets the minimum width constraint of the element.
MinHeight	Gets or sets the minimum height constraint of the element.
MaxWidth	Gets or sets the maximum width constraint of the element.
MaxHeight	Gets or sets the maximum height constraint of the element.
HorizontalContentAlignment	Gets or sets the horizontal alignment of the control's content. This is a dependency property.
VerticalContentAligment	Gets or sets the vertical alignment of the control's content. This is a dependency property.
C1Calendar.MonthCount	Gets or sets a number of months currently represented by the calendar. The default value is 1. This is a dependency property.
C1Calendar.MonthHeight	Determines a height of each month slot of the calendar.
C1Calendar.MonthWidth	Determines a width of each month slot of the calendar.

Multi-Month Calendar Display

The C1Calendar control can display multiple months with the ability to interactively navigate through months and select a specific DateTime or its components. C1Calendar builds its UI creating the necessary number of C1Calendar controls.



The most important properties responsible for the UI creation are:

- int C1Calendar.MonthCount – the number of months currently represented by the calendar;
- Style C1Calendar.MonthCalendarStyle – a Style applied to each child C1Calendar representing a single month;
- Style C1Calendar.MonthSlotStyle – a Style for C1CalendarPresenter elements which are the root elements of a visual tree representing a single month;
- ItemsPanelTemplate C1Calendar.MonthsPanel – an ItemsPanelTemplate that defines the panel that lays out elements representing separate months.

To provide a custom look for a C1Calendar:

1. To define a general layout model for a calendar, the **C1Calendar.Template** property should be assigned (usually through a Setter of a Style). The template visual tree should contain the C1CalendarPresenter to

designate a place where a panel with month calendars will appear.
2. To define the Months pane UI, assign a template to the C1Calendar.MonthsPanel property. This template will define a panel that lays out month items.
3. To define a single month UI, assign the Style to the C1Calendar.MonthSlotStyle property.

C1Calendar Alignment

You can specify where the calendar appears on the page using the **HorizontalAligment** and **VerticalAlignment** properties. In the **HorizontalAlignment** property you can select from Left, Center, Right, or Stretch. In the **VerticalAlignment** property you can select from Top, Center, Bottom, or Stretch. Using these properties gives you more flexibility in controlling the layout of your calendar(s) control.

C1Calendar Behavior

The following topics detail the behavior properties used to control the behavior of the C1Calendar control.

C1Calendar Navigation

C1Calendar provides interactive and simplified navigation for navigating calendar months and for navigating through calendar days. You can also use the C1Calendar control to navigate to the C1Scheduler control using the C1Calendar.BoldedDates and C1Calendar.SelectedDates properties.

C1Calendar includes the following methods for navigating through the calendar months:

- **Previous and Next** – Allows you to go to the previous or next month by clicking the button image or button image buttons.
- **Popup Calendar Month and Year Selector** – Clicking on the calendar month name opens up a listbox of calendar month names that enables you to select any month January-December to navigate to. Clicking on the calendar year date opens up a listbox of calendar year dates that enables you to select any year .

Note that regardless of what panel and day representing the item UI is used, the calendar provides the ability to interactively select a day (s) using the mouse or keyboard.

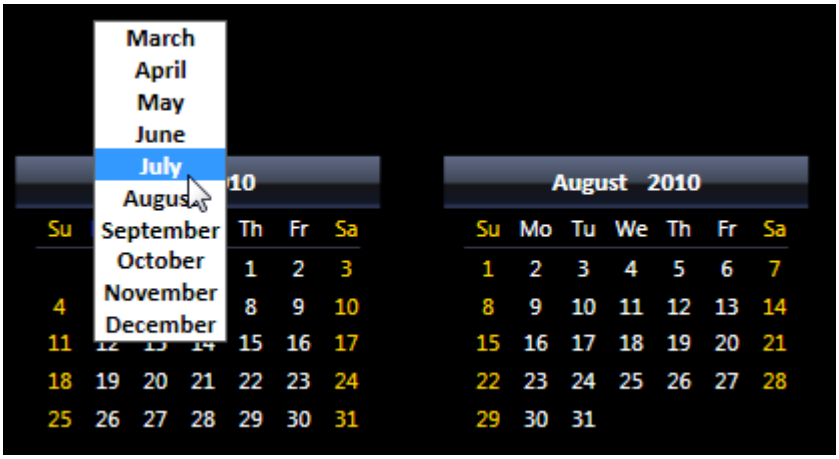
Previous and Next Navigation Buttons

Clicking the previous single arrow button navigates you to the previous month and clicking the next single arrow navigates you to the next month.

Popup Calendar Month and Year Selector

You can use the Quick Month Selector to jump to a particular calendar month without continuously clicking the navigation buttons. This is very helpful when you need to look ahead or back several calendar months.

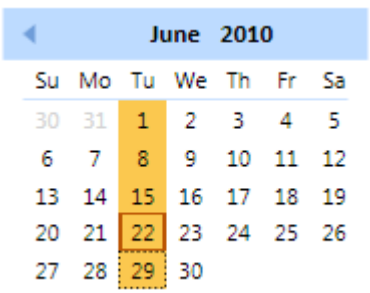
You can click the calendar month once to view calendar months. When you click once on the month name in the calendar title a popup listbox will appear with the names of the calendar months. Select the month you wish to navigate to and the calendar month will change from the current month to the selected month.



You can click the calendar year in the month title bar once to view calendar years. When you click once on the year in the calendar title a popup listbox will appear with the dates of the calendar year. Select the year you wish to navigate to and the calendar year will change from the current year to the selected year.

C1Calendar Selection

By default, one day can be selected at a time. You can select more than one day by increasing the value of the C1Calendar.MaxSelectionCount property. At run time depending on the value of the C1Calendar.MaxSelectionCount property, multiple days can be selected while holding the **CTRL** key and clicking the left mouse button. The selectable days appear with an orange backcolor by default.



To see how to set a maximum number of selectable days, see Specifying the Maximum Number of Days that can be Selected in C1Calendar.

You can specify the maximum and minimum date for date selection on the calendar through the CalendarBase.MaxDate and CalendarBase.MinDate properties. By default the minimum is 1/1/1753 and the maximum is 12/31/9998. When the maximum and minimum dates are applied, the dates past these values become unselectable. As a result, the selection color or action will not be applicable if the user attempts to select a date past the specified range.

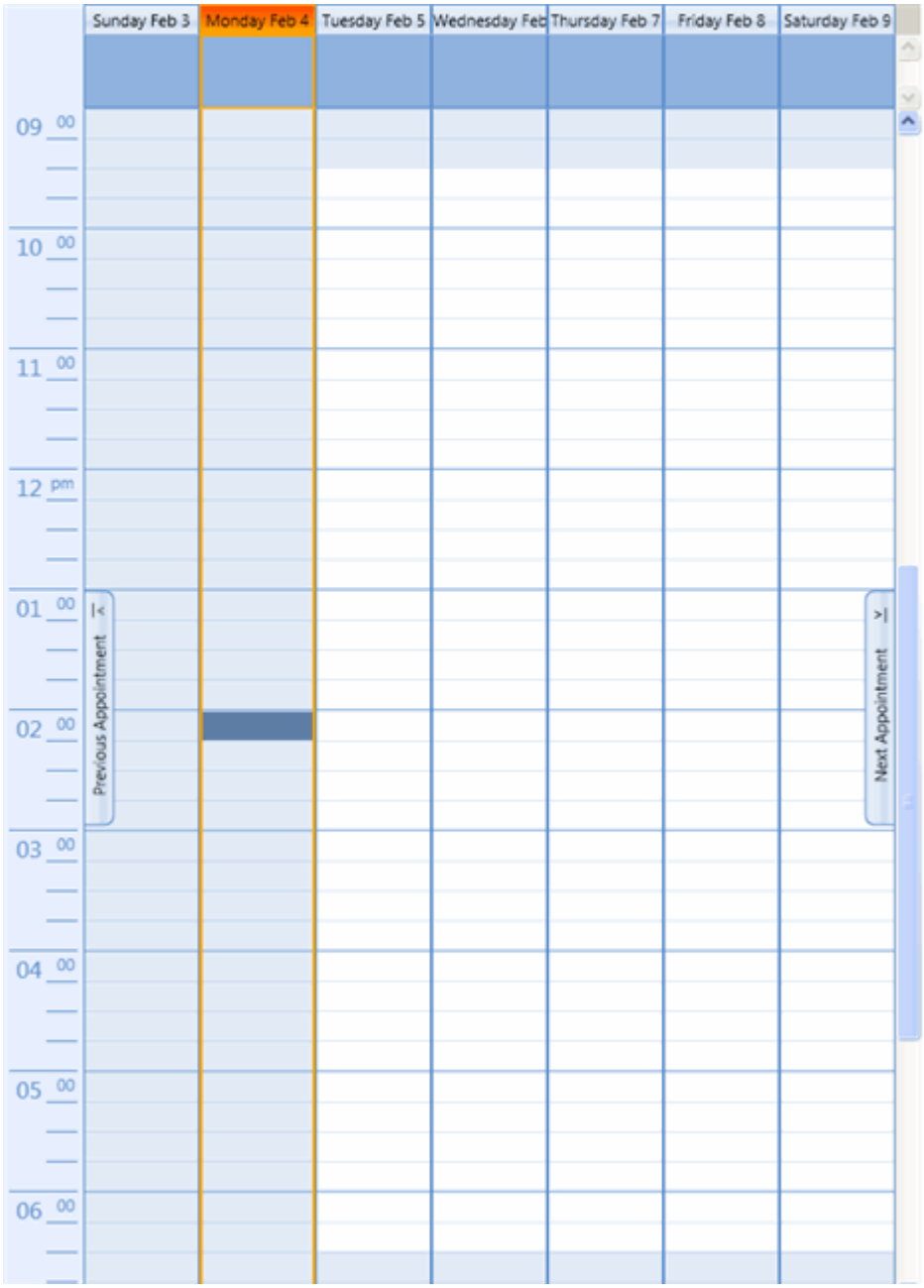
Settings

The C1Scheduler and C1Calendar controls share the same set of calendar settings. These settings are accessible via the corresponding properties of the CalendarHelper class. For example, you can create a work week calendar by specifying the work days through the CalendarHelper class. Notice that the XAML for the CalendarHelper class is the same for all three controls.

XAML for C1Scheduler

XAML

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



XAML for C1Calendar

XAML

```
<my:C1Calendar x:Name="scheduler1">
     <my:C1MonthCalendar.CalendarHelper>
           <my:CalendarHelper Culture="English " WeekStart="Sunday"
                EndDayTime="18:20:00" StartDayTime="09:20:00"
                WorkDays="Tuesday,Wednesday,Thursday,Friday,Saturday">
           </my:CalendarHelper>
     </my:C1MonthCalendar.CalendarHelper>
</my:C1MonthCalendar>
```

You can set C1Scheduler.CalendarHelper properties for one control and use them in several controls. This can be done by binding the C1Scheduler.CalendarHelper properties of corresponding controls to each other. For example, suppose

you have a C1Scheduler control with the previously specified C1Scheduler.CalendarHelper properties and a C1Calendar control with no properties specified yet in your window. Set the C1Calendar C1Scheduler.CalendarHelper properties as in the following XAML:

XAML

```
<my:C1Calendar Name="calendar1"
     CalendarHelper="{Binding ElementName=scheduler1, Path=CalendarHelper,
Mode=OneWay}"/>
```

The C1Calendar will display all of the C1Scheduler.CalendarHelper properties specified for the C1Scheduler control.

The following calendar settings are available:

CalendarHelper Property	Description
Culture	Gets or sets the **CultureInfo** object which holds culture-specific information.
WeekStart	Gets or sets the **DayOfWeek** value determining the first day of the week. The default is system settings.
WorkDays	Gets or sets the **WorkDays** object containing the set of working days in one week.
StartDayTime	Gets or sets the **TimeSpan** value specifying the beginning of the working time.
EndDayTime	Gets or sets the **TimeSpan** value specifying the end of the working time.
Holidays	Gets or sets **ObservableCollection<DateTime>** object which holds the list of holidays (non-working days in addition to weekends).
WeekendExceptions	Gets or sets the **ObservableCollection<DateTime>** object which holds the list of weekend days which should be working.
FullMonthNames	Gets an array of culture specific full month names.

Binding Schedule to a Calendar Control

The C1Calendar control can be used to provide navigation and additional information for a C1Scheduler control. This can be done by binding control properties.

C1MonthCalendar/C1MultiMonthCalendar Property	C1Scheduler Property	Comments
CalendarHelper	CalendarHelper	Binding these properties keeps the culture and calendar-dependant properties used by both controls in sync. This is important for default navigation.

VisibleDates	SelectedDates	Binding these properties allows the C1Scheduler control to be navigated when a user selects days in a calendar. For example, if you select four days in the C1Calendar, those four days will be shown in the C1Scheduler control.
BoldedDates	BoldedDates	Binding these properties makes days with appointments bold in a calendar control.

For example, the following XAML binds C1Scheduler to a C1Calendar control.

XAML

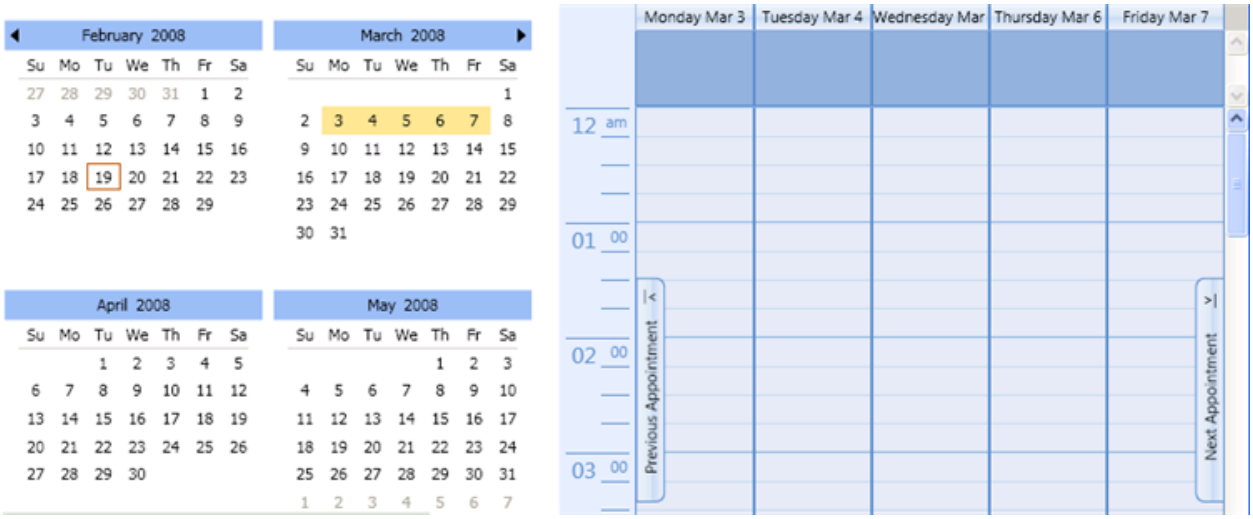
```
<Window
     xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
     xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
     xmlns:System="clr-namespace:System;assembly=mscorlib"
     x:Class="UntitledProject1.Window1"
     x:Name="Window"
     Title="Window1"
     Width="924" Height="480"
xmlns:c1sched="http://schemas.componentone.com/wpf/Schedule">

<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <c1:C1Scheduler Name="scheduler1" Grid.Column="1"
MonthStyle="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=MonthStyle2007}}"
WeekStyle="{DynamicResource {ComponentResourceKey
     TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=WeekStyle2007}}"
WorkingWeekStyle="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=WorkingWeekStyle2007}}"
OneDayStyle="{DynamicResource {ComponentResourceKey
     TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=OneDayStyle2007}}">
     <c1:C1Scheduler.CalendarHelper>
          <c1:CalendarHelper WeekStart="Sunday" EndDayTime="18:20:00"
StartDayTime="09:20:00"
               WorkDays="Tuesday,Wednesday,Friday,Saturday">
               <c1:CalendarHelper.Holidays>
                    <System:DateTime>2007-11-02</System:DateTime>
               <c1:CalendarHelper.Holidays>
          <c1:CalendarHelper>
     <c1:C1Scheduler.CalendarHelper>
  <c1:C1Scheduler>

  <c1:C1Calendar Name="calendar1" Margin="10,10,10,0"
```

```
Grid.Column="0" VerticalAlignment="Stretch" MaxSelectionCount="42"
     SelectedDates="{Binding ElementName=scheduler1, Path=VisibleDates, Mode=OneWay}"
     BoldedDates="{Binding ElementName=scheduler1, Path=BoldedDates, Mode=OneWay}"
     CalendarHelper="{Binding ElementName=scheduler1, Path=CalendarHelper,
Mode=OneWay}"
GenerateAdjacentMonthDays="true" >
     <c1:C1MultiMonthCalendar>
</Grid>
</Window>
```

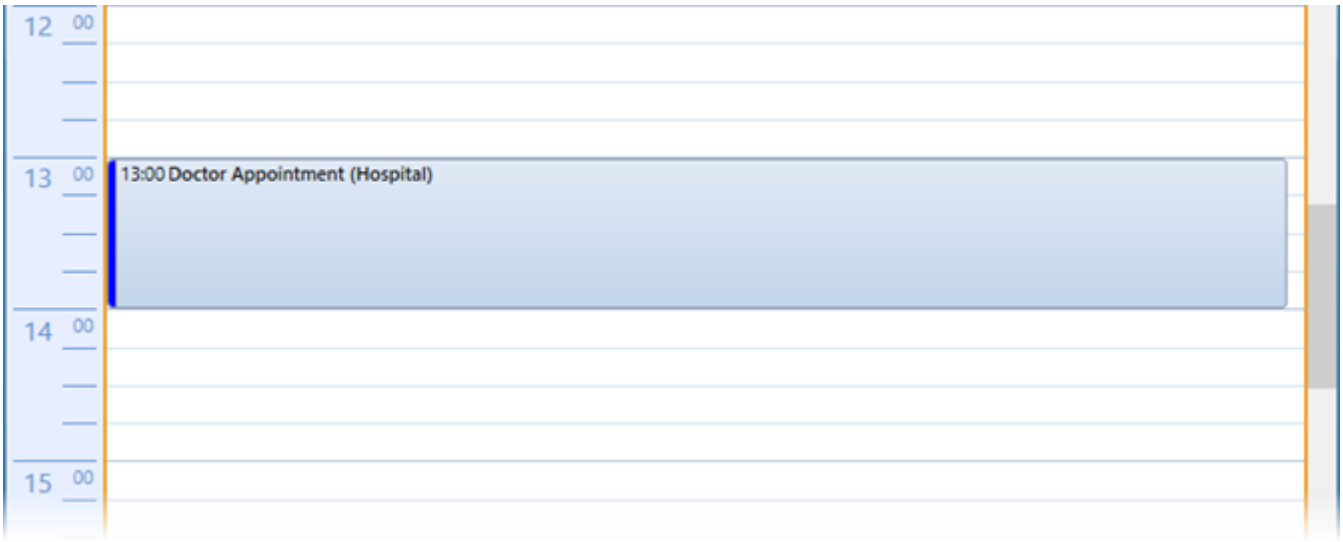
If several dates are selected in the C1Calendar, the C1Scheduler control changes to show the selected dates.



Appointments

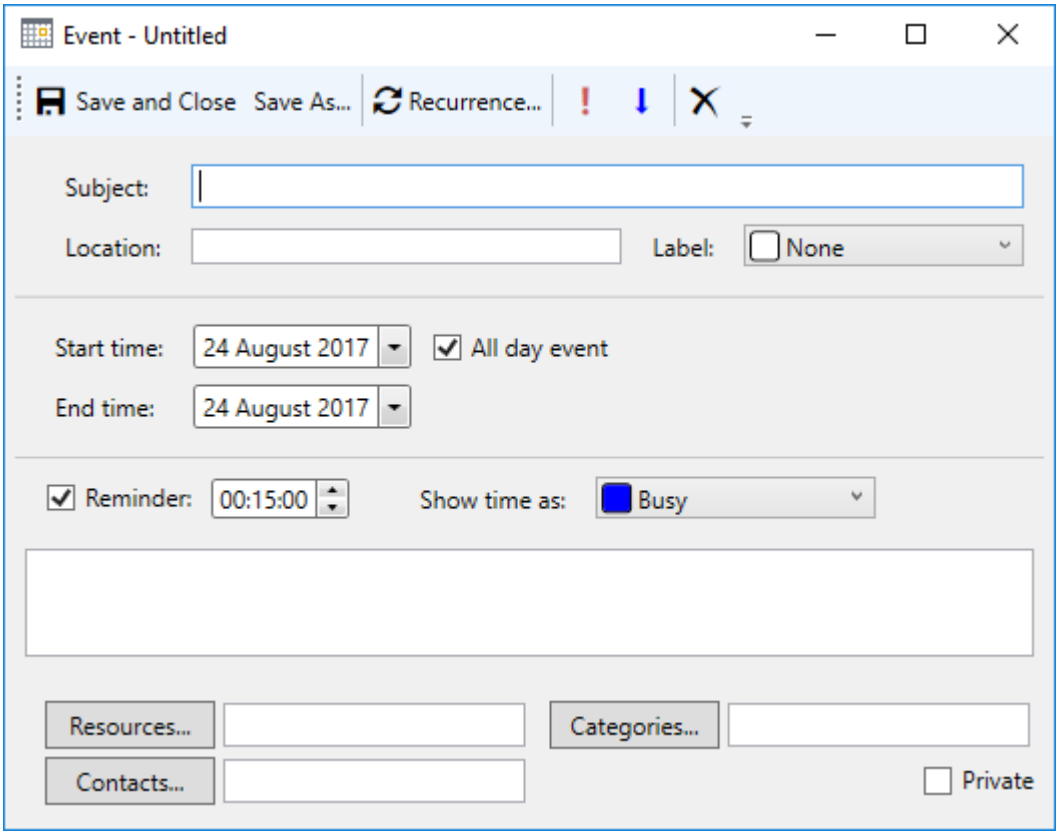
An appointment represents a period of time and detailed information about events that will take place during that period of time.

Appointments can span from a specified duration, such as 30 minutes, to multi-day events.



New appointments can be added or existing appointments can be edited by simply double-clicking the time of the appointment.

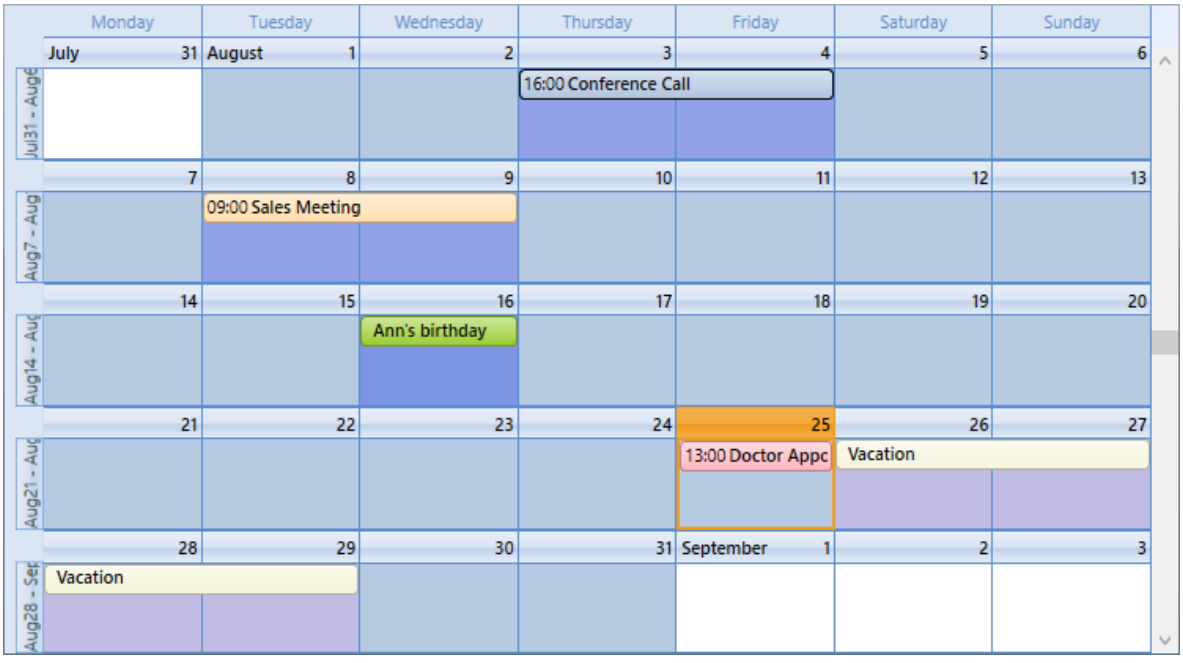
The **Appointment** dialog box is used to schedule new appointments, allowing you to set a subject, location, label, start and end time, reminder, availability status, and whether the appointment is an all day event and recurring over a specified period of time. You can also specify any resources, categories, and contacts here.



Labels

A label is simply a color-coded marker that can be added to appointments so that you and others know what type of appointment it is without even having to view its details.

There are twelve predefined labels available in **Scheduler for WPF and Silverlight**. The color of the label is visible in every data view in the C1Scheduler control.



Predefined Labels

The predefined labels include the following:

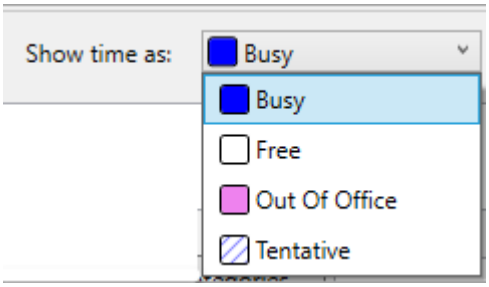
Label	Color	Index
None		0
Important		1
Business		2
Personal		3
Vacation		4
Deadline		5
Must Attend		6
Travel Required		7
Needs Preparation		8
Birthday		9
Anniversary		10
Phone Call		11

Availability Status

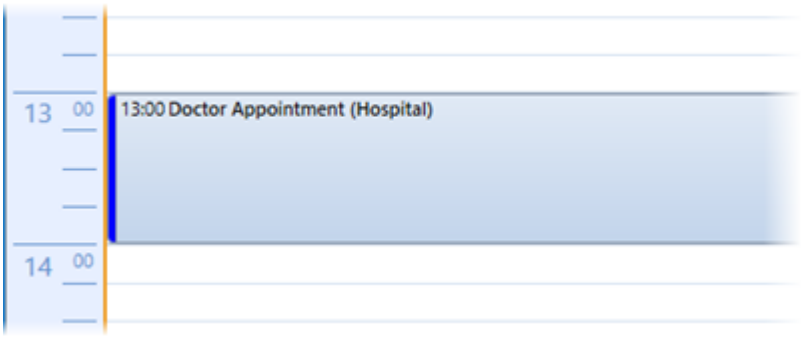
The availability status lets everyone know if you will be available at the time when an appointment is scheduled. There are four predefined availability statuses available in **Scheduler for WPF and Silverlight**: *Busy*, *Free*, *Out of Office*, and *Tentative*. The status is specified by the following colors:

Status	Color	Index
Busy		0
Free		1
Out of Office		2
Tentative		3

The status can be specified in the Appointment dialog box next to the **Show time as** drop-down list.

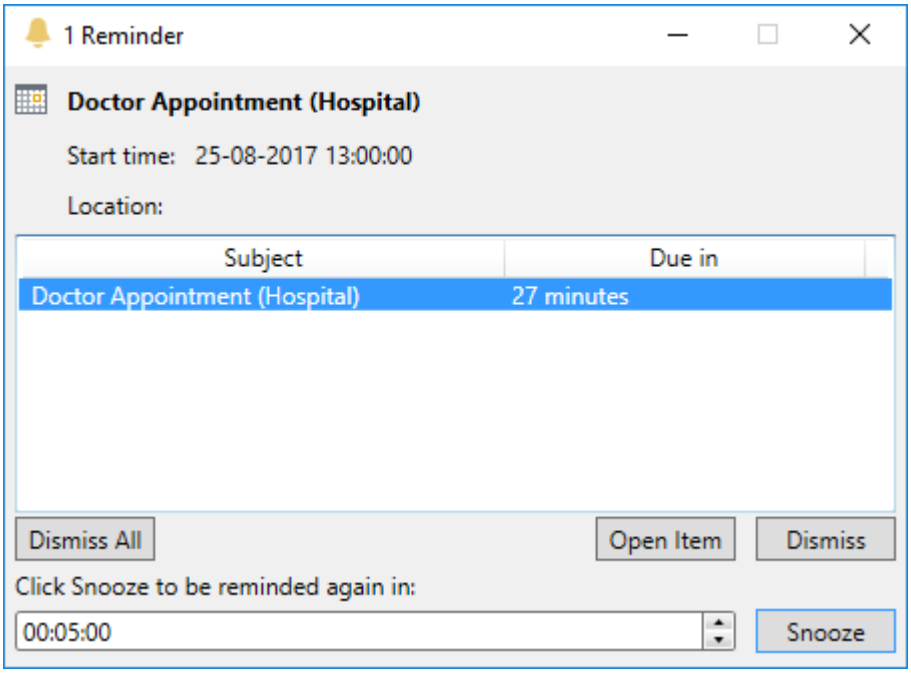


The color of the availability status is only visible in the **OneDayStyle** and **WorkingWeekStyle** views of a schedule. For appointments, the color appears in the area to the left of the appointment. Note that if you select the appointment, it will appear highlighted in the status color.

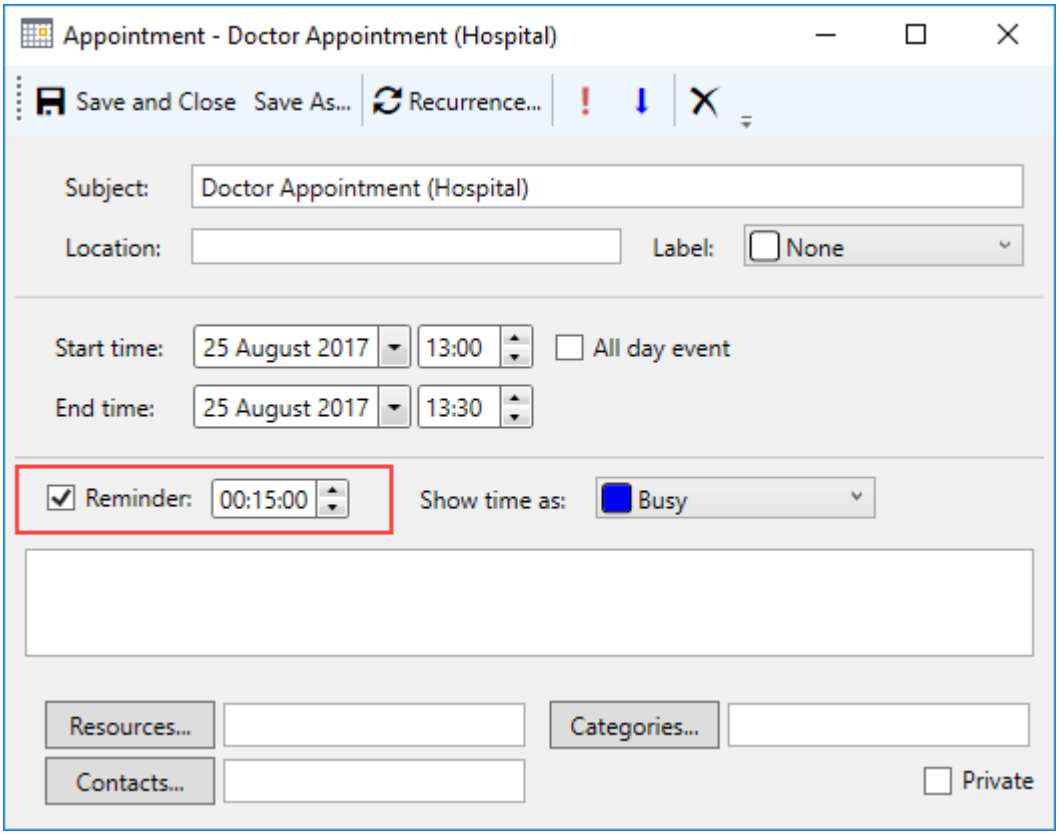


Reminders

Appointment reminders display a **Reminder** dialog box at a specified time before the appointment occurs. In the **Reminder** dialog box, you have the option of dismissing one or more appointments (if multiple appointments are due), opening the appointment, or resetting the reminder to appear again in a set amount of time.



Reminders can be set when creating an appointment by checking the **Reminder** check box and setting the amount of time before the appointment that you would like the reminder to appear in the **Reminder** drop-down list.



Contacts

A list of contacts usually includes all of the people with whom you frequently communicate. You might assign a contact to an appointment if this is the person you need to communicate with regarding the appointment. Contact information is stored in the ContactCollection class. Existing contacts can be retrieved from a data source, or you can add contacts to the **Contacts** at run time. Contacts are optional, and an appointment can have one or more contacts assigned to it.

Adding Contacts to the Contacts List

Scheduler for WPF and Silverlight supports contacts created at run time through the **Contacts** dialog box. Once added to the list, the contact can be assigned to an appointment.

To add a contact at run time:

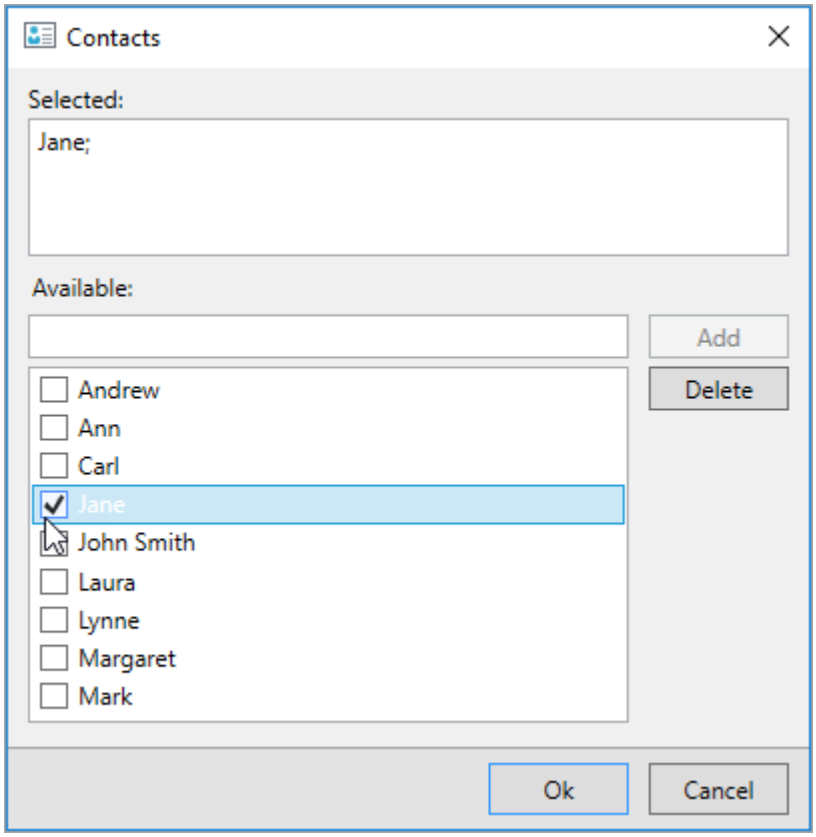
1. Add a new or open an existing appointment.
2. Click the **Contacts** button in the **Appointment** dialog box. The **Contacts** dialog box appears.
3. Enter a name in the **Available** text box and click **Add**.
4. Click **Ok** to close the **Contacts** dialog box.

Assigning Contacts to an Appointment

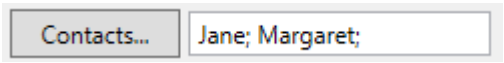
At run time, assigning a contact or contacts to an appointment can be done through the **Contacts** dialog box. For information on how to add items, see Adding Contacts to the Contacts List.

To assign a contact to an appointment at run time, follow these steps:

1. Click the **Contacts** button in the **Appointment** dialog box.
2. Select the check box next to the desired contact and click **Ok**.



The contact appears next in the **Contacts** text box. Note that you can add multiple contacts to an appointment.



Categories

A category is a keyword or a phrase used to help you organize your appointments. There are 20 predefined categories available in **Scheduler for WPF and Silverlight** to assign to appointments. You can also use categories from a database or users can create their own custom categories at run time. Categories, which are stored in the CategoryCollection class, are optional, and an appointment can have one or more categories assigned to it.

Predefined Categories

The predefined categories include the following:

Category	Index

Business	0
Competition	1
Favorites	2
Gifts	3
Goals/Objectives	4
Holiday	5
Holiday Cards	6
Hot Contacts	7
Ideas	8
International	9
Key Customer	10
Miscellaneous	11
Personal	12
Phone Calls	13
Status	14
Strategies	15
Suppliers	16
Time&Expenses	17
VIP	18
Waiting	19

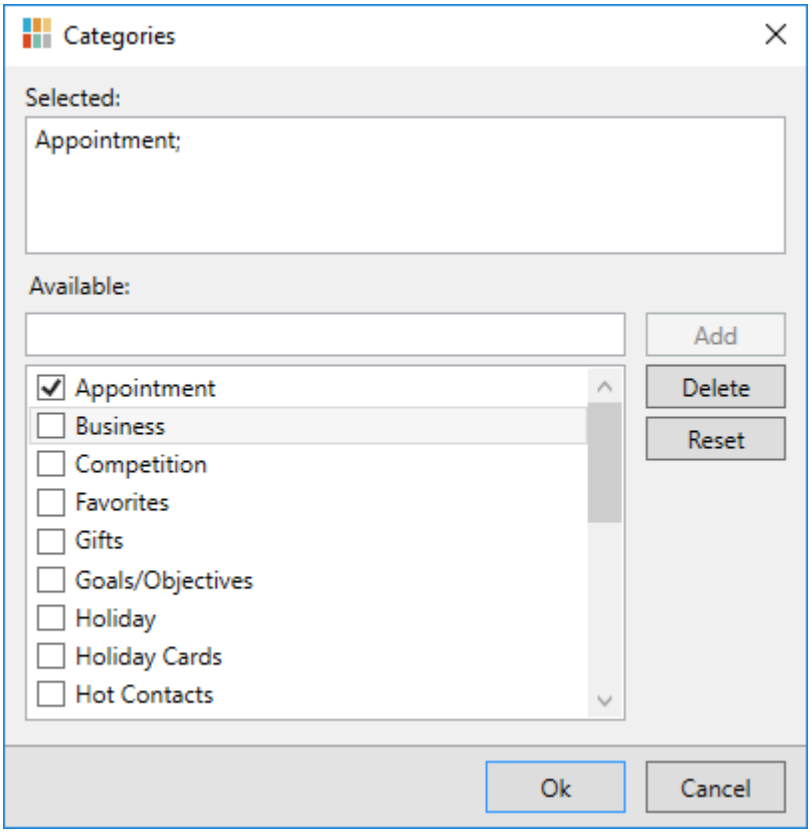
Adding Categories to the Categories List

Scheduler for WPF and Silverlight supports categories created at run time through the **Categories** dialog box. Once added to the list, the category can be assigned to an appointment.

To add a category at run time:

1. Add a new or open an existing appointment.
2. Click the **Categories** button in the **Appointment** dialog box. The **Categories** dialog box appears.
3. Enter a name in the **Available** text box and click **Add**. The new category is added at the end of the list.
4. Click **Ok** to close the **Categories** dialog box.

The new category appears in the **Categories** dialog box when the **Categories** button is clicked in the **Appointment** dialog box:

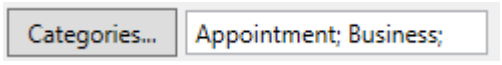


Assigning Categories to an Appointment

At run time, assigning a category or multiple categories to an appointment can be done through the **Categories** dialog box. By default, there is a list of twenty predefined categories. For more information on how to add categories to the **Categories** dialog box, see Adding Categories to the Categories List.

To assign a category to an appointment at run time:

1. Click the **Categories** button in the **Appointment** dialog box.
2. Select the check box next to the desired category and click **Ok**. The category appears next to the **Categories** text box. Note that you can assign multiple categories to an appointment.



Resources

A resource is a keyword or a phrase that defines a source of support for a particular task. Resources, which are stored in the **ResourceCollection** class, are optional and an appointment can have one or more resources assigned to it.

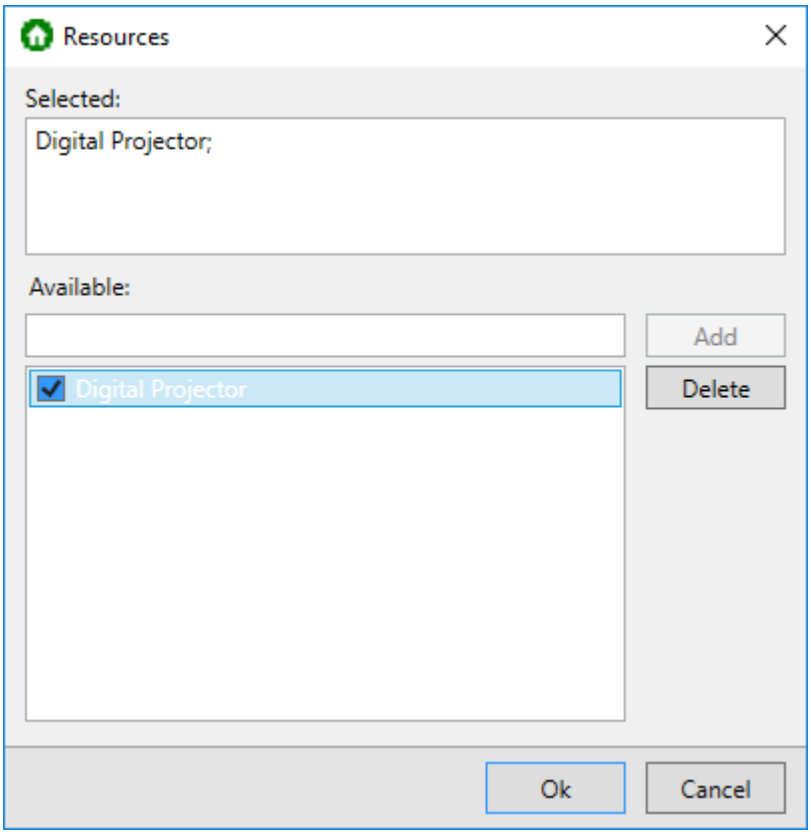
Adding Resources to the Resources List

Scheduler for WPF and Silverlight supports resources created at run time through the **Resources** dialog box. Once added to the list, the resource can be assigned to an appointment.

To add a resource at run time:

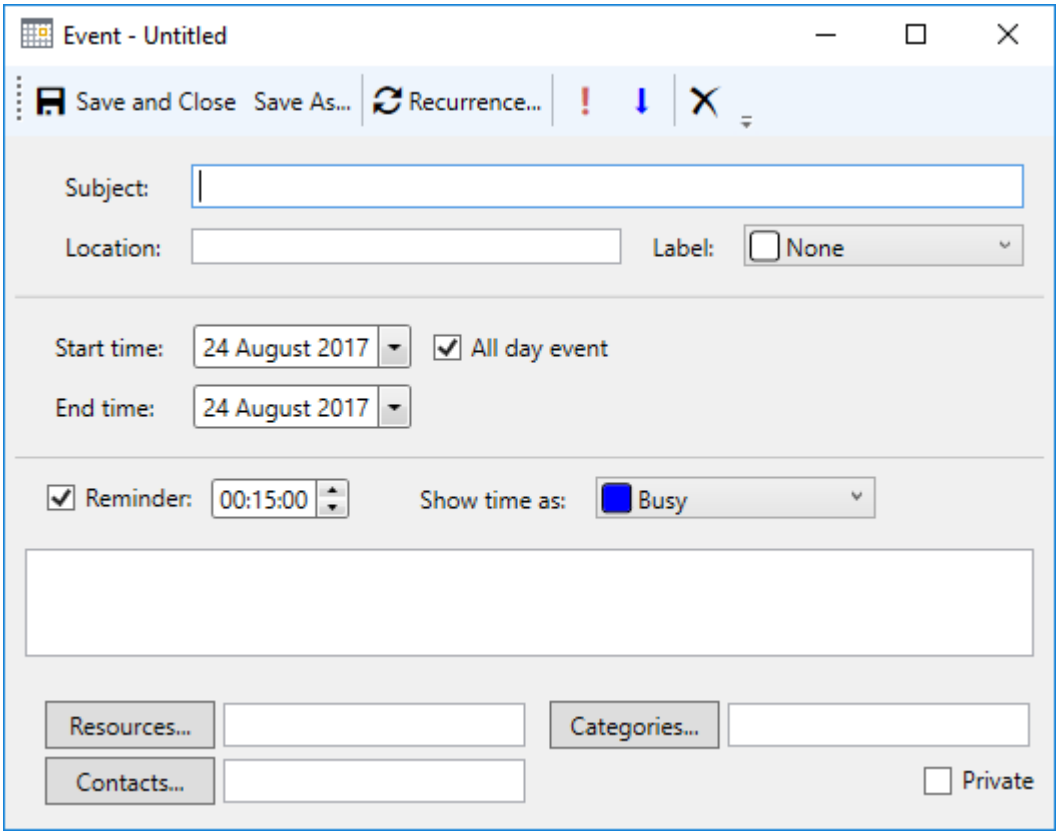
1. Add a new or open an existing appointment.
2. Click the **Resources** button in the **Appointment** dialog box. The **Resources** dialog box appears.
3. Enter a resource in the **Available** text box and click **Add**. The new resource is added to the list.
4. Click **Ok** to close the **Resources** dialog box.

The new resource appears in the **Resources** dialog box when the **Resources** button is clicked in the **Appointment** dialog box:



Working with Appointments at Run Time

Appointments can be created at run time by double-clicking the time for the appointment to begin or by pressing ENTER key, which will open the **Appointment** dialog box. For more information about appointments, see Appointments.



Scheduler for WPF and Silverlight 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

Adding and Saving an Appointment

Appointments can be added to the **Scheduler for WPF and Silverlight** using the **Appointment** dialog box.

1. Double-click the desired appointment time to open the **Appointment** dialog box, or simply press the ENTER key to create an in-place appointment.
2. In **Appointment dialog box,** specify a **Subject**, **Location** and any additional information you would like to assign to the appointment.
3. Click the **Save & Close** button to add the new appointment to the schedule.

Editing an Appointment

Appointments can be changed and updated in **Scheduler for WPF and Silverlight** either in-place or through the Appointment dialog box.

1. Use the F2 key to edit the Subject in-place. Press the Enter key to save edited text or Escape key to cancel editing.
2. Double-click an existing appointment to open the **Appointment** dialog box. Alternatively, use the Enter key to open the Appointment dialog box of the selected appointment.
3. After editing all the desired fields in the dialog use the **Save** button on the rightmost corner of the Appointment dialog box or Ctrl+S key to update the appointment in the schedule.

You can also move the selection and keyboard focus to the next or previous appointment in the current view by using the Tab (Shift+Tab) key on a selected appointment.

Deleting an Appointment

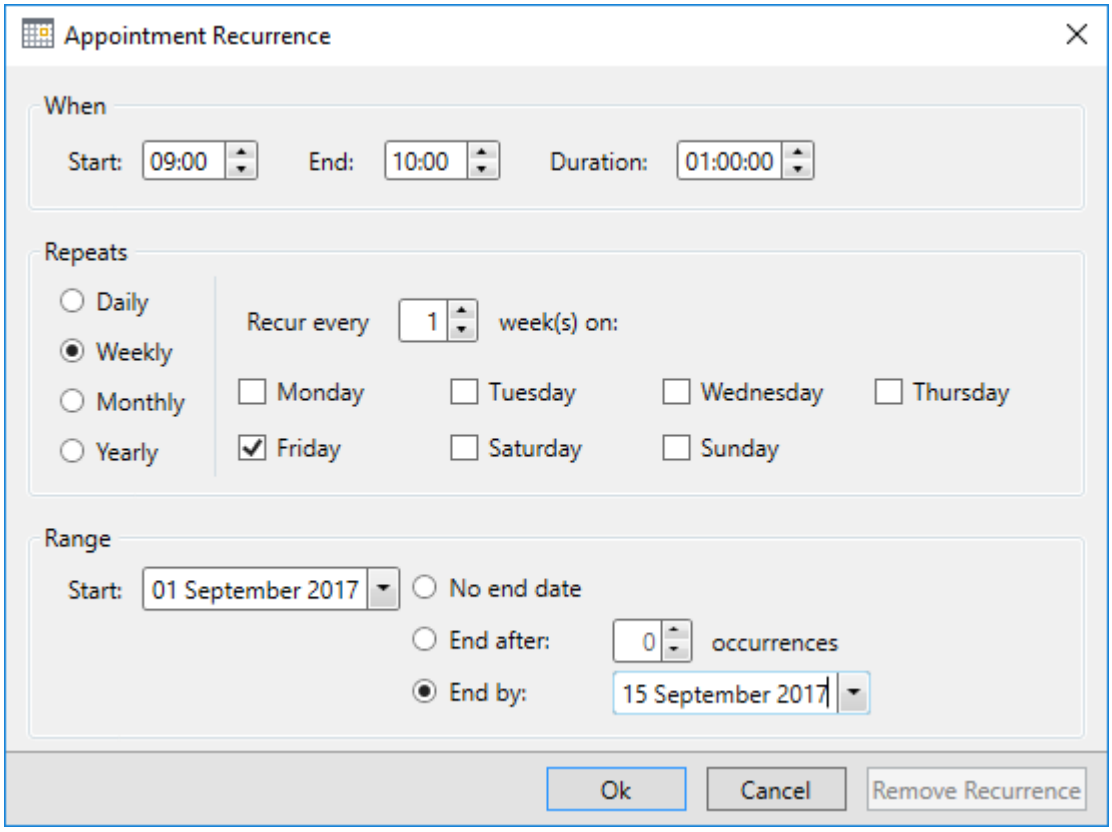
Selected appointments can be easily removed from your schedule through the **Edit Appointment** dialog box.

1. Select the appointment you would like to remove from the schedule.
2. Click the **Delete** key on your keyboard. The appointment is removed from the schedule.

Recurring Appointments

Appointments can be set to recur at specified intervals. An appointment can recur daily, weekly, monthly, or yearly.

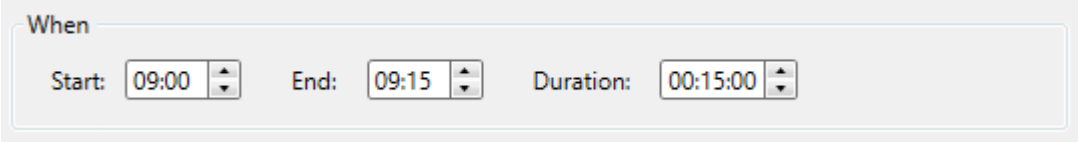
1. Double-click an existing appointment. The **Appointment** dialog box appears.
2. Click the **Recurrence** button. The **Appointment Recurrence** dialog box appears.



3. Set the desired recurrence pattern:

When

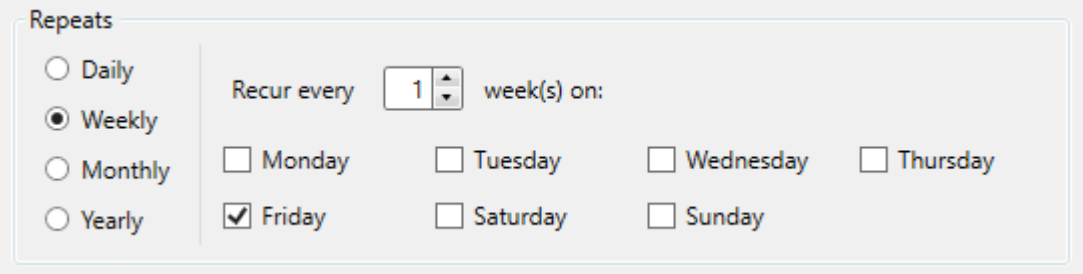
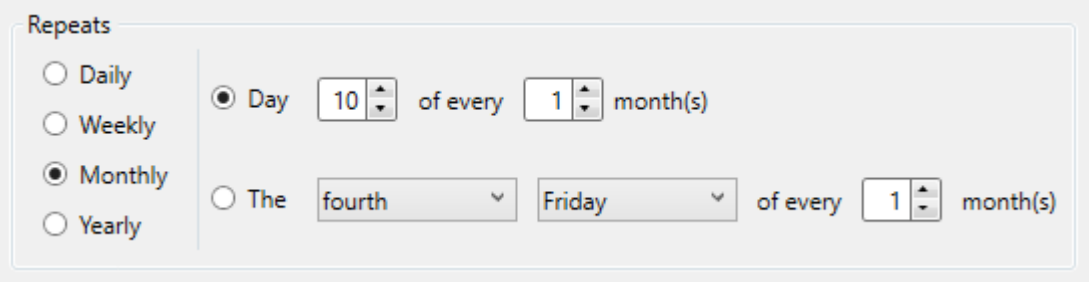
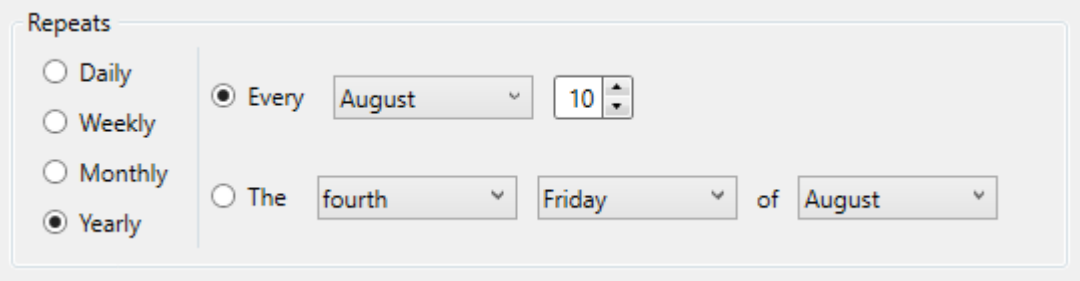
The properties in the **When** group allow you to set the start time, end time, and duration of the appointment.



Repeats

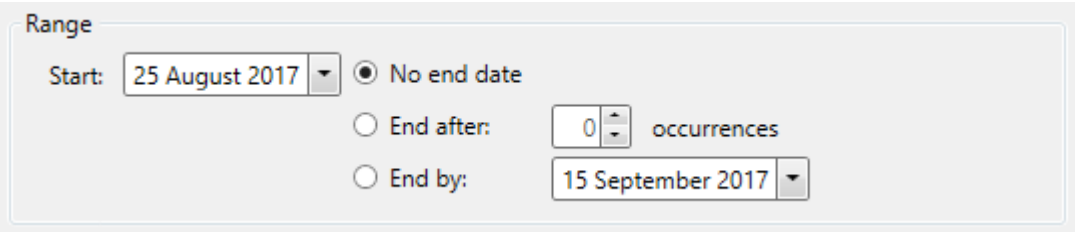
The **Repeats** group settings change depending on whether the appointment recurs **Daily**, **Weekly**, **Monthly**, or **Yearly**.

Daily	The **Daily** settings allow you to repeat an appointment every specified number of days or only on working days.  For example, setting the appointment to every 2 day(s) will make the appointment appear every other day. Setting the appointment to on working days will make the appointment

	appear only Monday through Friday, by default.
Weekly	The **Weekly** settings allow you to repeat the appointment on every specified week on specified days.  For example, setting the appointment to **every 2 week(s) on** and selecting **Tuesday** and **Thursday** will repeat the appointment every other week on Tuesdays and Thursdays.
Monthly	The **Monthly** settings allow you to repeat an appointment on every specified date at a specified interval of months or on a specified day and week of the month at a specified interval of months.  For example, setting the appointment to **on the 8 day of every 2 month** will make the appointment appear on the 8th of every other month. Setting the appointment to **on the third Monday of every 2 month** will make the appointment appear on the third Monday of every other month.
Yearly	The **Yearly** setting allows you to repeat an appointment on a specified date or a specified day and week of a specified month.  For example, setting the appointment to **on the 2 day of January** will make the appointment appear on January 2nd every year. This setting is ideal for birthdays and anniversaries. Setting the appointment to **on the first Friday of January** will make the appointment appear on the first Friday in January every year.

Range

The **Range** group allows you to set a time span for the recurrence.



The **Start** drop-down calendar allows you to select the date from which the recurrence will start. There are three options to choose from for an end date:

- **No end date** will make the appointment recur indefinitely.
- **End after 0 occurrences** will make the appointment recur a specified number of times. For example, if an appointment repeated every day, setting **End after 25 occurrences** would allow the appointment to repeat every day 25 times.
- **End by** will make the appointment recur until the date specified.

4. Click **Save** to close the **Appointment Recurrence** dialog box.

Scheduler for WPF Tutorials

The following tutorials assume that you are familiar with programming in Visual Basic .NET or C#. The tutorials provide step-by-step instructions; no prior knowledge of **C1Scheduler** is needed. By following the steps outlined in this section, you will be able to create projects demonstrating **C1Scheduler** features.

You are encouraged to run the tutorial projects, and experiment with your own modifications.

Creating a Custom Grouping View

C1Scheduler allows you to customize your Grouping View. This tutorial will walk you through creating the custom view through XAML markup and code.

Step 1 of 4: Creating the Application

In this step you will create the main Scheduler application using XAML markup and Code.

1. Create a new WPF application and add the following references by right-clicking **References** in the Solution Explorer and selecting **Add Reference** from the list:
    - C1.WPF.dll
    - C1.WPF.Schedule.dll
    - C1.WPF.DateTimeEditors.dll
2. Add the following XAML namespaces to your project in the <Window> tag:
    - xmlns:c1="clr-namespace:C1.WPF;assembly=C1.WPF"
    - xmlns:c1sched="clr-namespace:C1.WPF.Schedule;assembly=C1.WPF.Schedule"
    - xmlns:c1datetime="clr-namespace:C1.WPF.DateTimeEditors;assembly=C1.WPF.DateTimeEditors"

    The <*Window*> tag should resemble the following:

XAML

```
<Window x:Class="CustomGroupingView.MainWindow"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="Custom Grouping View"
  xmlns:c1sched="http://schemas.componentone.com/wpf/Schedule"
xmlns:c1datetime="http://schemas.componentone.com/wpf/DateTimeEditors"
xmlns:c1="http://schemas.componentone.com/wpf/Basic"
  Width="1024" Height="800">
```

3. Add a set of <*Window.Resources*> </*Window.Resources*> tags above the <*Grid*> tags.
4. Insert the following markup between the <*Window.Resources*> </*Window.Resources*> tags to create your Resource Dictionary:

XAML

```
<ResourceDictionary>
            <ResourceDictionary x:Key="{ComponentResourceKey
TypeInTargetAssembly={x:Type c1sched:C1Scheduler},
```

```
                            ResourceId=custom_theme}" Source="CustomViews.xaml"
/>
        </ResourceDictionary>
```

5. Insert the following XAML after the *<Window.Resources> </Window.Resources>* tags to create the schedule view and the C1Scheduler control:

XAML

```
<Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition />
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="192"/>
            <ColumnDefinition />
        </Grid.ColumnDefinitions>
        <ToolBar Grid.Row="0" Grid.ColumnSpan="2">
            <TextBlock HorizontalAlignment="Left" VerticalAlignment="Center"
TextWrapping="NoWrap" Margin="4,2" Text="Contacts navigation:" />
            <Button Content="|&lt;&lt;" Margin="2"

Command="c1sched:C1Scheduler.NavigateToPreviousGroupCommand"
                                CommandParameter="Home" CommandTarget="{Binding
ElementName=c1Scheduler1}"/>
            <Button Content="&lt;&lt;" Margin="2"

Command="c1sched:C1Scheduler.NavigateToPreviousGroupCommand"
                                CommandParameter="Page" CommandTarget="{Binding
ElementName=c1Scheduler1}"/>
            <Button Content="&lt;" Margin="2"

Command="c1sched:C1Scheduler.NavigateToPreviousGroupCommand"
CommandTarget="{Binding ElementName=c1Scheduler1}"/>
            <Button Content="&gt;" Margin="2"

Command="c1sched:C1Scheduler.NavigateToNextGroupCommand" CommandTarget="{Binding
ElementName=c1Scheduler1}"/>
            <Button Content="&gt;&gt;" Margin="2"

Command="c1sched:C1Scheduler.NavigateToNextGroupCommand"
                                CommandParameter="Page" CommandTarget="{Binding
ElementName=c1Scheduler1}"/>
            <Button Content="&gt;&gt;|" Margin="2"

Command="c1sched:C1Scheduler.NavigateToNextGroupCommand"
                                CommandParameter="End" CommandTarget="{Binding
ElementName=c1Scheduler1}"/>
            <Separator/>
```

```
            <TextBlock HorizontalAlignment="Left" VerticalAlignment="Center"
TextWrapping="NoWrap" Margin="4,2" Text="Contacts per page:" />
            <c1:C1NumericBox Margin="2" Value="{Binding GroupPageSize,
ElementName=c1Scheduler1, Mode=TwoWay}" Minimum="2" Maximum="5" MinWidth="35"/>
        </ToolBar>
        <c1sched:C1Calendar x:Name="cal1" Grid.Row="1" Grid.Column="0"
                            CalendarHelper="{Binding CalendarHelper,
ElementName=c1Scheduler1, Mode=OneWay}"
                                    SelectedDates="{Binding VisibleDates,
ElementName=c1Scheduler1, Mode=OneWay}"
                                        BoldedDates="{Binding BoldedDates,
ElementName=c1Scheduler1, Mode=OneWay}"
                            MaxSelectionCount="42" />
        <c1sched:C1Scheduler x:Name="c1Scheduler1" Grid.Row="1" Grid.Column="1"
Style="{Binding WorkingWeekStyle, ElementName=c1Scheduler1}"
             Theme="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=custom_theme}}">
            <c1sched:C1Scheduler.Settings>
                <c1sched:C1SchedulerSettings FirstVisibleTime="7:00:00" />
            </c1sched:C1Scheduler.Settings>
        </c1sched:C1Scheduler>
    </Grid>
```

In this step you created a the markup that will call the Resource Dictionary and created a custom Scheduler view. In the next step, you will add code to your project to handle adding contacts and returning the VisualIntervalCollection for specified days.

Step 2 of 4: Adding Code to the Application

In this step you will add code to the application to handle adding contacts and to control returning the VisualIntervalCollection for a specified day.

1. Right-click on **MainPage.xaml** and select **View Code** from the list.
2. Add the following namespaces to your application:

Visual Basic

```
Imports C1.C1Schedule
Imports C1.WPF.Schedule
Imports C1.WPF
Imports System.Collections
```

C#

```
using C1.C1Schedule;
using C1.WPF.Schedule;
using C1.WPF;
using System.Collections;
```

3. Insert the following code directly below the **InitializeComponent()** method:

Visual Basic

```
' add some contacts
   Dim cnt As New Contact()
   cnt.Text = "Andy Garcia"
   c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt)
   cnt = New Contact()
   cnt.Text = "Nancy Drew"
   c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt)
   cnt = New Contact()
   cnt.Text = "Robert Clark"
   c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt)
   cnt = New Contact()
   cnt.Text = "James Doe"
   c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt)
   c1Scheduler1.GroupBy = "Contact"
```

C#

```
// add some contacts
    Contact cnt = new Contact();
    cnt.Text = "Andy Garcia";
    c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt);
    cnt = new Contact();
    cnt.Text = "Nancy Drew";
    c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt);
    cnt = new Contact();
    cnt.Text = "Robert Clark";
    c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt);
    cnt = new Contact();
    cnt.Text = "James Doe";
    c1Scheduler1.DataStorage.ContactStorage.Contacts.Add(cnt);
    c1Scheduler1.GroupBy = "Contact";
```

4. The following code returns the VisualIntervalCollection for the specified day and SchedulerGroup:

Visual Basic

```
''' <summary>
   ''' Returns VisualIntervalCollection for the specified day and specified
SchedulerGroup which can be used
   ''' as an ItemsSource for VisualIntervalsPresenter control.
   ''' </summary>
   ''' <remarks>
   ''' If converter parameter is "Self", return list of a single
VisualIntervalGroup, to use it as ItemsSource for representing all-day area.
   ''' In all other cases returns VisualItervalCollection containing time slots
```

```
for the single day.
    ''' </remarks>
    Public Class GroupItemToVisualIntervalsConverter
        Implements IValueConverter
        Public Shared [Default] As New GroupItemToVisualIntervalsConverter()
        #Region "IValueConverter Members"
        Public Function Convert(ByVal value As Object, ByVal targetType As
Type, ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object Implements IValueConverter.Convert
            Dim el As FrameworkElement = TryCast(value, FrameworkElement)
            If el IsNot Nothing Then
                Dim group As SchedulerGroupItem = TryCast(el.DataContext,
SchedulerGroupItem)
                Dim index As Integer = -1
                If group IsNot Nothing Then
                    Dim itm As ItemsControl =
TryCast(VTreeHelper.GetParentOfType(el, GetType(ItemsControl)), ItemsControl)
                    If itm IsNot Nothing Then
                        Dim data As Object = itm.DataContext
                        Dim itmParent As ItemsControl =
TryCast(VTreeHelper.GetParentOfType(itm, GetType(ItemsControl)), ItemsControl)
                        If itmParent IsNot Nothing Then
                            index = itmParent.Items.IndexOf(data)
                            Dim visualIntervalGroup As
VisualIntervalGroup = TryCast(group.VisualIntervalGroups(index),
VisualIntervalGroup)
                            Dim param As String = CStr(parameter)
                            If param.ToLower() = "self" Then
                                ' create list of a single
VisualIntervalGroup
                                ' (we need list to use it as
ItemsSource)
                                Dim list As New List(Of Object)()
                                list.Add(visualIntervalGroup)
                                Return list
                            Else
                                Return
visualIntervalGroup.VisualIntervals
                            End If
                        End If
                    End If
                End If
                Return Nothing
        End Function
        Public Function ConvertBack(ByVal value As Object, ByVal targetType As
Type, ByVal parameter As Object, ByVal culture As
System.Globalization.CultureInfo) As Object Implements
IValueConverter.ConvertBack
            Throw New NotImplementedException()
        End Function
```

```
        #End Region
   End Class
End Namespace
```

C#

```
/// <summary>
    /// Returns VisualIntervalCollection for the specified day and specified
SchedulerGroup which can be used
    /// as an ItemsSource for VisualIntervalsPresenter control.
    /// </summary>
    /// <remarks>
    /// If converter parameter is "Self", return list of a single
VisualIntervalGroup, to use it as ItemsSource for representing all-day area.
    /// In all other cases returns VisualItervalCollection containing time slots
for the single day.
    /// </remarks>
    public class GroupItemToVisualIntervalsConverter : IValueConverter
    {
        public static GroupItemToVisualIntervalsConverter Default = new
GroupItemToVisualIntervalsConverter();
        #region IValueConverter Members
        public object Convert(object value, Type targetType, object parameter,
System.Globalization.CultureInfo culture)
        {
            FrameworkElement el = value as FrameworkElement;
            if (el != null)
            {
                SchedulerGroupItem group = el.DataContext as SchedulerGroupItem;
                int index = -1;
                if (group != null)
                {
                    ItemsControl itm = VTreeHelper.GetParentOfType(el,
typeof(ItemsControl)) as ItemsControl;
                    if (itm != null)
                    {
                        object data = itm.DataContext;
                        ItemsControl itmParent =
VTreeHelper.GetParentOfType(itm, typeof(ItemsControl)) as ItemsControl;
                        if (itmParent != null)
                        {
                            index = itmParent.Items.IndexOf(data);
                            VisualIntervalGroup visualIntervalGroup =
group.VisualIntervalGroups[index] as VisualIntervalGroup;
                            string param = (string)parameter;
                            if (param.ToLower() == "self")
                            {
                                // create list of a single VisualIntervalGroup
                                // (we need list to use it as ItemsSource)
                                List<object> list = new List<object>();
```

```
                                list.Add(visualIntervalGroup);
                                return list;
                            }
                            else
                            {
                                return visualIntervalGroup.VisualIntervals;
                            }
                        }
                    }
                }
            }
            return null;
        }
        public object ConvertBack(object value, Type targetType, object
parameter, System.Globalization.CultureInfo culture)
        {
            throw new NotImplementedException();
        }

        #endregion
    }
}
```

In this step you added code to create contacts and to return VisualIntervalCollections for the specified day. In the next step you will create your Resource Dictionary.

Step 3 of 4: Creating the Resource Dictionary

In this step, you will create the Resource Dictionary for your application.

1. Create a Resource Dictionary by right-clicking the application name and selecting **Add | New Item** from the list.
2. Select **Resource Dictionary** from the list of installed templates and name it **CustomViews.xaml**. Click **OK**.
3. Replace the markup on the page with the following namespace declaration:

XAML

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:sys="clr-namespace:System;assembly=mscorlib"
    xmlns:c1="clr-namespace:C1.WPF;assembly=C1.WPF"
    xmlns:c1sched="clr-namespace:C1.WPF.Schedule;assembly=C1.WPF.Schedule"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:CustomGroupingView"

xmlns:PresentationOptions="http://schemas.microsoft.com/winfx/2006/xaml/presentation/options">
</ResourceDictionary>
```

4. Insert the following XAML after the namespace declarations:

XAML

```
<ResourceDictionary.MergedDictionaries>
        <ResourceDictionary
Source="/C1.WPF.Schedule;component/themes/SchedulerThemes/Office2007/Default.xaml" />
```

```
    </ResourceDictionary.MergedDictionaries>
```

5. Use the following XAML markup to create a Data Template:

XAML

```
<!-- determines the template used for displaying group headers -->
    <DataTemplate x:Key="GroupHeaderTemplate">
        <Grid SnapsToDevicePixels="True">
            <c1:C1BrushBuilder x:Name="Background" Style="{StaticResource
C1Scheduler_ControlArea_BrushStyle}" Input="{Binding Background}"/>
            <c1:C1BrushBuilder x:Name="BorderBrush" Style="{StaticResource
C1Scheduler_MonthHeaderForeground_BrushStyle}" Input="{Binding Path=Scheduler.Background}"/>
            <Border VerticalAlignment="Stretch" HorizontalAlignment="Stretch"
                        BorderThickness="0,0,1,0" BorderBrush="{Binding Output,
ElementName=BorderBrush}"
                        Background="{Binding Output, ElementName=Background}">
                <TextBlock Foreground="{Binding Path=Scheduler.Foreground}" Margin="0"
TextWrapping="Wrap"
                           Text="{Binding DisplayName}" VerticalAlignment="Center"
HorizontalAlignment="Center"/>
            </Border>
        </Grid>
    </DataTemplate>
```

6. Insert the following XAML beneath the Data Template to determine the style of the day group in a day view:

XAML

```
<!-- determines the style of the day group in a day view -->
    <Style TargetType="ItemsControl" x:Key="DayGroup_Style">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ItemsControl">
                    <Grid>
                        <c1:C1BrushBuilder x:Name="BorderBrush" Style="{StaticResource
C1Scheduler_Border_Style}"
                                           Input="{Binding
Path=DataContext.Scheduler.Background, RelativeSource={RelativeSource TemplatedParent}}"/>
                        <Border BorderThickness="1,0,0,0" x:Name="border"
                                BorderBrush="{Binding ElementName=BorderBrush, Path=Output}">
                            <ItemsPresenter />
                        </Border>
                    </Grid>
                    <ControlTemplate.Triggers>
                        <DataTrigger Binding="{Binding Path=StartTime, Mode=OneTime,
Converter={x:Static c1sched:IsTodayConverter.Default}}" Value="True">
                            <Setter TargetName="border" Property="BorderBrush"
                                    Value="{Binding
Path=DataContext.Scheduler.TodayBackground, RelativeSource={RelativeSource TemplatedParent}}"
/>
                            <Setter TargetName="border" Property="BorderThickness"
Value="2,0,1,0" />
                        </DataTrigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
```

```
    </Style>
```

7. The XAML below creates the style used for displaying time slots in a Day view:

XAML

```
<!-- determines the style used for displaying time slots in a Day view.
         Copy it into every theme, as using DynamicResource here affects performance -->
    <c1sched:TimeSlotStyleSelector x:Key="TimeSlotStyleSelector">
        <!-- Don't use selection states. With selection states multiple selection works
incorrectly. -->
        <c1sched:TimeSlotStyleSelector.Resources>
            <ResourceDictionary>
                <Style x:Key="C1Scheduler_WorkSlot_Style"
TargetType="c1sched:VisualIntervalPresenter">
                    <Setter Property="BorderBrush" Value="{StaticResource
C1Scheduler_WorkHourLightBorder_Brush}" />
                    <Setter Property="Template">
                        <Setter.Value>
                            <ControlTemplate TargetType="c1sched:VisualIntervalPresenter">
                                <Grid>
                                    <c1:C1BrushBuilder x:Name="BorderBrush"
Style="{StaticResource C1Scheduler_Border_Style}"
                                                       Input="{Binding Path=Scheduler.Background}"/>
                                    <Border BorderThickness="0,0,1,0" x:Name="border"
IsHitTestVisible="False"
                                                         BorderBrush="{Binding ElementName=BorderBrush,
Path=Output}">
                                        <Border Background="{Binding
Path=Scheduler.AlternatingBackground}"
                                                          BorderThickness="0,1,0,0"
                                                          BorderBrush="{TemplateBinding BorderBrush}">
                                            <Border Background="{Binding Tag, RelativeSource=
{RelativeSource AncestorType={x:Type c1sched:VisualIntervalsPresenter}}}"
                                                            Visibility="{Binding IsSelected, Converter=
{x:Static c1sched:BooleanToVisibilityConverter.Default}}"/>
                                        </Border>
                                    </Border>
                                    <ContentPresenter Content="{TemplateBinding Content}"
Margin="0,1,1,0"
                                                          ContentTemplate="{TemplateBinding
ContentTemplate}" />
                                </Grid>
                                <ControlTemplate.Triggers>
                                    <DataTrigger Binding="{Binding Path=StartTime,
Mode=OneTime, Converter={x:Static c1sched:IsTodayConverter.Default}}" Value="True">
                                        <Setter TargetName="border" Property="BorderBrush"
                                            Value="{Binding Path=Scheduler.TodayBackground}"
/>
                                    </DataTrigger>
                                </ControlTemplate.Triggers>
                            </ControlTemplate>
                        </Setter.Value>
                    </Setter>
                </Style>

                <Style x:Key="C1Scheduler_TopWorkSlot_Style"
                       BasedOn="{StaticResource C1Scheduler_WorkSlot_Style}"
```

```
                                TargetType="c1sched:VisualIntervalPresenter">
                    <Setter Property="BorderBrush" Value="{StaticResource
C1Scheduler_WorkHourBorder_Brush}" />
                </Style>

                <Style x:Key="C1Scheduler_FreeSlot_Style"
TargetType="c1sched:VisualIntervalPresenter">
                    <Setter Property="BorderBrush" Value="{StaticResource
C1Scheduler_FreeHourLightBorder_Brush}" />
                    <Setter Property="Template">
                        <Setter.Value>
                            <ControlTemplate TargetType="c1sched:VisualIntervalPresenter">
                                <Grid>
                                    <c1:C1BrushBuilder x:Name="BorderBrush"
Style="{StaticResource C1Scheduler_Border_Style}"
                                                       Input="{Binding Path=Scheduler.Background}"/>
                                    <Border BorderThickness="0,0,1,0" x:Name="border"
IsHitTestVisible="False"
                                                        BorderBrush="{Binding ElementName=BorderBrush,
Path=Output}">
                                        <Border Background="{StaticResource
C1Scheduler_FreeHour_Brush}"
                                                BorderThickness="0,1,0,0"
                                                BorderBrush="{TemplateBinding BorderBrush}">
                                            <Border Background="{Binding Tag, RelativeSource=
{RelativeSource AncestorType={x:Type c1sched:VisualIntervalsPresenter}}}"
                                                    Visibility="{Binding IsSelected, Converter=
{x:Static c1sched:BooleanToVisibilityConverter.Default}}"/>
                                        </Border>
                                    </Border>
                                    <ContentPresenter Content="{TemplateBinding Content}"
Margin="0,1,1,0"
                                                      ContentTemplate="{TemplateBinding
ContentTemplate}" />
                                </Grid>

                                <ControlTemplate.Triggers>
                                    <DataTrigger Binding="{Binding Path=StartTime,
Mode=OneTime, Converter={x:Static c1sched:IsTodayConverter.Default}}" Value="True">
                                        <Setter TargetName="border" Property="BorderBrush"
                                            Value="{Binding Path=Scheduler.TodayBackground}"
/>
                                    </DataTrigger>
                                </ControlTemplate.Triggers>
                            </ControlTemplate>
                        </Setter.Value>
                    </Setter>
                </Style>

                <Style x:Key="C1Scheduler_TopFreeSlot_Style" BasedOn="{StaticResource
C1Scheduler_FreeSlot_Style}"
                       TargetType="c1sched:VisualIntervalPresenter">
                    <Setter Property="BorderBrush" Value="{StaticResource
C1Scheduler_FreeHourBorder_Brush}" />
                </Style>

            </ResourceDictionary>
        </c1sched:TimeSlotStyleSelector.Resources>
    </c1sched:TimeSlotStyleSelector>
```

8. Determine the style used for displaying the All-Day area in a Day view:

XAML

```
<!-- determines the style used for displaying All-Day area in a Day view -->
    <Style x:Key="C1Scheduler_AllDayInterval_Style"
TargetType="c1sched:VisualIntervalPresenter">
        <Setter Property="BorderThickness" Value="0,1,0,2" />
        <Setter Property="SnapsToDevicePixels" Value="True"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="c1sched:VisualIntervalPresenter">
                    <Grid>
                        <c1:C1BrushBuilder x:Name="BackgroundBrush" Style="{StaticResource
C1Scheduler_AllDayArea_Style}"
                                           Input="{Binding Path=OwnerGroup.Background}"/>
                        <c1:C1BrushBuilder x:Name="SelectedBrush" Style="{StaticResource
C1Scheduler_AllDayAreaSelected_Style}"
                                           Input="{Binding Path=OwnerGroup.Background}"/>
                        <c1:C1BrushBuilder x:Name="BorderBrush" Style="{StaticResource
C1Scheduler_Border_Style}"
                                           Input="{Binding Path=Scheduler.Background}"/>
                        <Border x:Name="AllDayBorder" IsHitTestVisible="False"
                                Background="{Binding ElementName=BackgroundBrush,
Path=Output}"
                                BorderThickness="{TemplateBinding BorderThickness}"
                                BorderBrush="{Binding Tag, RelativeSource={RelativeSource
AncestorType={x:Type c1sched:VisualIntervalsPresenter}}}">
                            <Border x:Name="AllDayBorder1" BorderThickness="0,0,1,0"
                                    BorderBrush="{Binding Path=Output,
ElementName=BorderBrush}">
                                <Border Background="{Binding ElementName=SelectedBrush,
Path=Output}"
                                                                    Visibility="{Binding IsSelected,
Converter={x:Static c1sched:BooleanToVisibilityConverter.Default}}"/>
                            </Border>
                        </Border>
                        <Border Background="{Binding Path=VisualIntervals[0].StatusBrush}"
Opacity="0.2"
                                Margin="{Binding BorderThickness, ElementName=AllDayBorder}"
                                c1sched:CoverElementsPane.Orientation="Horizontal"
                                c1sched:CoverElementsPane.PaneName="allDayPane" />
                    </Grid>
                    <ControlTemplate.Triggers>
                        <DataTrigger Binding="{Binding Path=StartTime, Mode=OneTime,
Converter={x:Static c1sched:IsTodayConverter.Default}}" Value="True">
                            <Setter TargetName="AllDayBorder" Property="BorderBrush"
Value="{Binding Path=Scheduler.TodayBackground}" />
                            <Setter TargetName="AllDayBorder1" Property="BorderBrush"
Value="{Binding Path=Scheduler.TodayBackground}" />
                        </DataTrigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

9. Define the style for your next/previous appointment pane:

XAML

```
<!-- Defines a style for previous/next appointment navigation pane (containing
            next/previous labels) represented by ContentControl. -->
    <Style x:Key="PrevNextAppPane_Style" TargetType="ContentControl">
        <Setter Property="Focusable" Value="False" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="ContentControl">
                    <Grid x:Name="nextPrevAppGrid" SnapsToDevicePixels="True">
                        <Grid.RowDefinitions>
                            <RowDefinition />
                            <RowDefinition Height="Auto" />
                            <RowDefinition />
                        </Grid.RowDefinitions>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition Width="Auto" />
                            <ColumnDefinition />
                            <ColumnDefinition Width="Auto" />
                        </Grid.ColumnDefinitions>
                        <c1:C1BrushBuilder x:Name="BackgroundBrush" Style="{StaticResource
C1Scheduler_NavPane_Style}" Input="{TemplateBinding Background}"/>
                        <c1:C1BrushBuilder x:Name="BorderBrush" Style="{StaticResource
C1Scheduler_Border_Style}" Input="{TemplateBinding Background}"/>
                        <c1:C1BrushBuilder x:Name="MouseOverBrush" Style="{StaticResource
C1Scheduler_NavPane_HoverStyle}" Input="{TemplateBinding Background}"/>
                        <Button Grid.Row="1" Grid.Column="0"
                                Content="{Binding PreviousAppointmentText}"
                                CommandTarget="{Binding DataContext, RelativeSource=
{RelativeSource TemplatedParent}}"

Command="c1sched:C1Scheduler.NavigateToPreviousAppointmentCommand"
                                Foreground="{Binding DataContext.Foreground, RelativeSource=
{RelativeSource TemplatedParent}}"
                                Background="{Binding Output, ElementName=BackgroundBrush}"
                                BorderBrush="{Binding Output, ElementName=BorderBrush}"
                                Tag="{Binding Output, ElementName=MouseOverBrush}"
                                Style="{StaticResource C1Scheduler_PreviousButton_Style}">
                            <Button.LayoutTransform>
                                <RotateTransform Angle="-90" />
                            </Button.LayoutTransform>
                        </Button>
<!--  DataContext="{TemplateBinding DataContext}"/>-->
                        <Button Grid.Row="1" Grid.Column="2"
                                Content="{Binding NextAppointmentText}"
                                CommandTarget="{Binding DataContext, RelativeSource=
{RelativeSource TemplatedParent}}"

Command="c1sched:C1Scheduler.NavigateToNextAppointmentCommand"
                                Foreground="{Binding DataContext.Foreground, RelativeSource=
{RelativeSource TemplatedParent}}"
                                Background="{Binding Output, ElementName=BackgroundBrush}"
                                BorderBrush="{Binding Output, ElementName=BorderBrush}"
                                Tag="{Binding Output, ElementName=MouseOverBrush}"
                                Style="{StaticResource C1Scheduler_NextButton_Style}">
                            <Button.LayoutTransform>
                                <RotateTransform Angle="-90" />
                            </Button.LayoutTransform>
```

```
                        </Button>
                        <!--  DataContext="{TemplateBinding DataContext}"/>-->
                    </Grid>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>
```

10. The following XAML will set that your application will retrieve from the Resource Dictionary:

XAML

```
<!-- Style that represents One Day View -->
    <Style x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type c1sched:C1Scheduler},
                  ResourceId=OneDayStyle}" TargetType="c1sched:C1Scheduler"
BasedOn="{StaticResource BaseViewStyle}">
        <Style.Resources>
            <Style TargetType="{x:Type ListBoxItem}">
                <Setter Property="ListBoxItem.Padding" Value="0" />
            </Style>
        </Style.Resources>
        <Setter Property="c1sched:C1Scheduler.VisualIntervalScale" Value="00:30:00" />
        <Setter Property="ShowWorkTimeOnly" Value="false" />
        <Setter Property="c1sched:C1Scheduler.Template">
            <Setter.Value>
                <ControlTemplate TargetType="c1sched:C1Scheduler">
                    <Border Background="{Binding RelativeSource={RelativeSource
TemplatedParent}, Path=ControlBackground}"
                            BorderBrush="{Binding RelativeSource={RelativeSource
TemplatedParent}, Path=BorderBrush}"
                            BorderThickness="{Binding RelativeSource={RelativeSource
TemplatedParent}, Path=BorderThickness}">
                        <DockPanel SnapsToDevicePixels="True">
                            <!-- DayHeader grid (includes day headers) -->
                            <Grid DockPanel.Dock="Top" >
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition Width="57" />
                                    <ColumnDefinition />
                                    <ColumnDefinition Width="17"/>
                                </Grid.ColumnDefinitions>
                                <!-- hidden SchedulerPresenters for each group (they should
generate VisualIntervals) -->
                                <ItemsControl Height="0"
                                              ItemsSource="{Binding RelativeSource=
{RelativeSource TemplatedParent}, Path=VisibleGroupItems}">
                                    <ItemsControl.ItemTemplate>
                                        <DataTemplate>
                                            <c1sched:C1SchedulerPresenter />
                                        </DataTemplate>
                                    </ItemsControl.ItemTemplate>
                                </ItemsControl>
                                <c1:C1BrushBuilder x:Name="DayHeaderBorderBrush"
                                              Style="{StaticResource
C1Scheduler_Border_Style}"
                                              Input="{Binding
Path=Scheduler.Background}"/>
                                <ItemsControl Grid.Column="1" Tag="{Binding Output,
ElementName=DayHeaderBorderBrush}"
                                              ItemsSource="{Binding RelativeSource=
```

```
{RelativeSource TemplatedParent}, Path=VisibleGroupItems[0].VisualIntervalGroups}">
                                    <ItemsControl.ItemsPanel>
                                        <ItemsPanelTemplate>
                                            <UniformGrid Rows="1" />
                                        </ItemsPanelTemplate>
                                    </ItemsControl.ItemsPanel>
                                    <ItemsControl.ItemTemplate>
                                        <DataTemplate>
                                            <Grid>
                                                <Grid.RowDefinitions>
                                                    <RowDefinition Height="Auto"/>
                                                    <RowDefinition Height="Auto"/>
                                                </Grid.RowDefinitions>
                                                <Button x:Name="dayHeaderButton" Grid.Row="0"
MinHeight="25" Padding="0,2,0,2"

Content="{Binding Converter={x:Static c1sched:VisualIntervalToStringConverter.Default},

ConverterParameter=DayViewDayHeaderFormat}"

Style="{DynamicResource C1Scheduler_DayHeaderButton_Style}"

CommandParameter="{Binding Path=Scheduler.OneDayStyle}"

Command="c1sched:C1Scheduler.ChangeStyleCommand"/>
                                                <ItemsControl x:Name="GroupHeaderList"
Grid.Row="1" HorizontalAlignment="Stretch" Focusable="False"
                                                              Style="{StaticResource
DayGroup_Style}"
                                                              Visibility="{Binding
Path=Scheduler.IsGrouped, Converter={x:Static c1sched:BooleanToVisibilityConverter.Default}}"
                                                              ItemsSource="{Binding
Path=Scheduler.VisibleGroupItems}" Background="{Binding Path=OwnerGroup.Background}">
                                                    <ItemsControl.ItemTemplate>
                                                        <DataTemplate>
   <!-- Group Header  -->
                                                            <ContentControl  MinHeight="25"

HorizontalContentAlignment="Stretch" VerticalContentAlignment="Stretch"

ContentTemplate="{Binding Scheduler.GroupHeaderTemplate}"

Content="{Binding}"/>
                                                        </DataTemplate>
                                                    </ItemsControl.ItemTemplate>
                                                    <ItemsControl.ItemsPanel>
                                                        <ItemsPanelTemplate>
                                                            <UniformGrid Rows="1" />
                                                        </ItemsPanelTemplate>
                                                    </ItemsControl.ItemsPanel>
                                                </ItemsControl>
                                            </Grid>
                                        </DataTemplate>
                                    </ItemsControl.ItemTemplate>
                                </ItemsControl>
                                <Border Grid.Column="2" />
                            </Grid>
                            <!-- AllDay area grid (should scroll vertically) -->
                            <ScrollViewer DockPanel.Dock="Top" Height="54" BorderThickness="0"
Padding="0"
```

```
                                    Focusable="False"
                                HorizontalScrollBarVisibility="Disabled"
VerticalScrollBarVisibility="Visible">
                                <Grid>
                                    <Grid.ColumnDefinitions>
                                        <ColumnDefinition Width="57" />
                                        <ColumnDefinition />
                                    </Grid.ColumnDefinitions>
                                    <ItemsControl Grid.Column="1"

ItemsSource="{Binding RelativeSource={RelativeSource TemplatedParent},

Path=VisibleGroupItems[0].VisualIntervalGroups}">
                                        <ItemsControl.ItemsPanel>
                                            <ItemsPanelTemplate>
                                                <UniformGrid Rows="1" />
                                            </ItemsPanelTemplate>
                                        </ItemsControl.ItemsPanel>
                                        <ItemsControl.ItemTemplate>
                                            <DataTemplate>
                                                <Grid>
                                                    <ItemsControl ItemsSource="{Binding
Path=Scheduler.VisibleGroupItems}"
                                                                  Style="{StaticResource
DayGroup_Style}">
                                                        <ItemsControl.ItemsPanel>
                                                            <ItemsPanelTemplate>
                                                                <UniformGrid Rows="1"/>
                                                            </ItemsPanelTemplate>
                                                        </ItemsControl.ItemsPanel>
                                                        <ItemsControl.ItemTemplate>
                                                            <DataTemplate>
                                                                <Grid>
                                                                    <c1:C1BrushBuilder
x:Name="AllDayBorderBrush"

Style="{StaticResource C1Scheduler_Border_Style}"
                                                                                       Input="{Binding
Background}"/>

<c1sched:VisualIntervalGroupsPresenter

HorizontalAlignment="Stretch" VerticalAlignment="Stretch"
                                                                                       Tag="{Binding
Output, ElementName=AllDayBorderBrush}"

Background="{Binding Background}"

ItemsSource="{Binding RelativeSource={RelativeSource Self},
                                                                                            Converter=
{x:Static local:GroupItemToVisualIntervalsConverter.Default},

ConverterParameter=Self}"

ItemContainerStyle="{DynamicResource C1Scheduler_AllDayInterval_Style}">

<ItemsControl.ItemsPanel>

<ItemsPanelTemplate>
```

```
                                                                    <UniformGrid
Rows="1"/>
</ItemsPanelTemplate>
</ItemsControl.ItemsPanel>
</c1sched:VisualIntervalGroupsPresenter>
<c1sched:AppointmentsCoverPane x:Name="allDayPane" Owner="{Binding Owner}"
UseSimpleLayout="false"
c1sched:AppointmentsCoverPane.AppointmentFilter="Event"
IsDragDropDisabled="{Binding Path=Scheduler.IsDragDropDisabled}"
ExtendOnOverflow="True" CoverElementsMargin="10">
<c1sched:AppointmentsCoverPane.Resources>
<ResourceDictionary>
                                                                    <Thickness
x:Key="TimeBorderThickness">0</Thickness>
</ResourceDictionary>
</c1sched:AppointmentsCoverPane.Resources>
</c1sched:AppointmentsCoverPane>
                                                        </Grid>
                                                    </DataTemplate>
                                                </ItemsControl.ItemTemplate>
                                            </ItemsControl>
                                        </Grid>
                                    </DataTemplate>
                                </ItemsControl.ItemTemplate>
                            </ItemsControl>
                        </Grid>
                    </ScrollViewer>
                    <!-- TimeSlots grid (should scroll vertically) -->
                    <ScrollViewer x:Name="scrollViewer" BorderThickness="0"
Padding="0" Focusable="False" MaxHeight="100000"
                                  VerticalScrollBarVisibility="Visible"
HorizontalScrollBarVisibility="Disabled">
                        <Grid>
                            <Grid.ColumnDefinitions>
                                <ColumnDefinition Width="57" />
                                <ColumnDefinition />
                            </Grid.ColumnDefinitions>
                            <ItemsControl Grid.Column="0" x:Name="HourMarkersList"
                                    ItemsSource="{Binding RelativeSource=
{RelativeSource TemplatedParent},
Path=VisibleGroupItems[0].VisualIntervalGroups[0].VisualIntervals}"
                                    ItemTemplate="{DynamicResource
C1Scheduler_TimeRuler_Template}">
                                <ItemsControl.ItemsPanel>
                                    <ItemsPanelTemplate>
                                        <UniformGrid Columns="1" />
```

```
                                        </ItemsPanelTemplate>
                                    </ItemsControl.ItemsPanel>
                                </ItemsControl>
                                <ItemsControl Grid.Column="1"
ItemsSource="{Binding RelativeSource={RelativeSource TemplatedParent},
Path=VisibleGroupItems[0].VisualIntervalGroups}">
                                    <ItemsControl.ItemsPanel>
                                        <ItemsPanelTemplate>
                                            <UniformGrid Rows="1" />
                                        </ItemsPanelTemplate>
                                    </ItemsControl.ItemsPanel>
                                    <ItemsControl.ItemTemplate>
                                        <DataTemplate>
                                            <Grid>
                                                <ItemsControl ItemsSource="{Binding
Path=Scheduler.VisibleGroupItems}" Style="{StaticResource DayGroup_Style}">
                                                    <ItemsControl.ItemsPanel>
                                                        <ItemsPanelTemplate>
                                                            <UniformGrid Rows="1"/>
                                                        </ItemsPanelTemplate>
                                                    </ItemsControl.ItemsPanel>
                                                    <ItemsControl.ItemTemplate>
                                                        <DataTemplate>
                                                            <Grid>
                                                                <c1:C1BrushBuilder
x:Name="selectedBackground"
Style="{DynamicResource C1Scheduler_TimeSlotSelected_BrushStyle}"
Input="{Binding Background}"/>
<c1sched:VisualIntervalsPresenter x:Name="presenter"
ItemContainerStyleSelector="{DynamicResource TimeSlotStyleSelector}"
ItemsSource="{Binding RelativeSource={RelativeSource Self},
                                                                                    Converter=
{x:Static local:GroupItemToVisualIntervalsConverter.Default},
ConverterParameter=VisualIntervals}"
ItemTemplate="{Binding Path=Scheduler.VisualIntervalTemplate}"
Background="{Binding Background}"
                                                                                Tag="{Binding
Output, ElementName=selectedBackground}">
<c1sched:VisualIntervalsPresenter.ItemsPanel>
<ItemsPanelTemplate>
                                                                                    <UniformGrid
Columns="1"/>
</ItemsPanelTemplate>
</c1sched:VisualIntervalsPresenter.ItemsPanel>
</c1sched:VisualIntervalsPresenter>
```

```
<c1sched:AppointmentsCoverPane x:Name="appPane" Owner="{Binding Owner}"
UseSimpleLayout="false" SizingType="Proportional"
c1sched:AppointmentsCoverPane.AppointmentFilter="Appointment"
IsDragDropDisabled="{Binding Path=Scheduler.IsDragDropDisabled}"
CoverElementsMargin="10">
<c1sched:AppointmentsCoverPane.Resources>
<ResourceDictionary>
                                                                                <Thickness
x:Key="TimeBorderThickness">5,0,0,0</Thickness>
</ResourceDictionary>
</c1sched:AppointmentsCoverPane.Resources>
</c1sched:AppointmentsCoverPane>
                                                                    </Grid>
                                                                </DataTemplate>
                                                            </ItemsControl.ItemTemplate>
                                                        </ItemsControl>
                                                    </Grid>
                                                </DataTemplate>
                                            </ItemsControl.ItemTemplate>
                                        </ItemsControl>
                                        <!-- next/previous appointment navigation pane -->
                                        <Canvas Grid.Column="1" x:Name="cnv" >
                                            <ContentControl Canvas.Left="0"
Canvas.Top="{Binding VerticalOffset, ElementName=scrollViewer}"
                                                            DataContext="{Binding RelativeSource=
{RelativeSource TemplatedParent}}"
Width="{Binding ActualWidth, ElementName=cnv}"
Height="{Binding ViewportHeight, ElementName=scrollViewer}"
Background="{Binding Background, RelativeSource={RelativeSource TemplatedParent}}"
Style="{DynamicResource PrevNextAppPane_Style}"
Visibility="{Binding HasVisibleAppointments, RelativeSource={RelativeSource TemplatedParent},
                                                    Converter={x:Static
c1sched:BooleanToVisibilityConverter.Default}, ConverterParameter=Invert}"/>
                                        </Canvas>
                                    </Grid>
                                </ScrollViewer>
                            </DockPanel>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="c1sched:C1Scheduler.VisualIntervalGroupDescriptions">
                <Setter.Value>
                    <c1sched:IntervalGroupDescriptionCollection>
```

```
                    <c1sched:VisualIntervalGroupDescription PropertyName="StartTime.Day" />
                </c1sched:IntervalGroupDescriptionCollection>
            </Setter.Value>
        </Setter>
        <Setter Property="c1sched:C1Scheduler.VisualIntervalPanel">
            <Setter.Value>
                <ItemsPanelTemplate>
                    <UniformGrid Columns="1" />
                </ItemsPanelTemplate>
            </Setter.Value>
        </Setter>
        <Setter Property="c1sched:C1Scheduler.VisualTimeSpan" Value="1" />
        <Setter Property="c1sched:C1Scheduler.VisualIntervalTemplate">
            <Setter.Value>
                <DataTemplate DataType="{x:Type c1sched:VisualInterval}">
                    <!-- don't place C1BrushBuilder here, it affects performance, use binding
to the relative ancestor instead -->
                    <Border Background="{Binding Path=StatusBrush}" Opacity="0.2"

c1sched:CoverElementsPane.Orientation="Vertical"
                                                    c1sched:CoverElementsPane.PaneName="appPane"
MinHeight="20"/>
                </DataTemplate>
            </Setter.Value>
        </Setter>
    </Style>

    <Style x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type c1sched:C1Scheduler},
             ResourceId=WeekStyle}" TargetType="c1sched:C1Scheduler"
           BasedOn="{StaticResource {ComponentResourceKey TypeInTargetAssembly={x:Type
c1sched:C1Scheduler},
             ResourceId=OneDayStyle}}">
        <Setter Property="VisualTimeSpan" Value="7" />
    </Style>

    <Style x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type c1sched:C1Scheduler},
             ResourceId=WorkingWeekStyle}" TargetType="c1sched:C1Scheduler"
           BasedOn="{StaticResource {ComponentResourceKey TypeInTargetAssembly={x:Type
c1sched:C1Scheduler},
             ResourceId= OneDayStyle}}">
        <Setter Property="VisualTimeSpan" Value="7" />
    </Style>

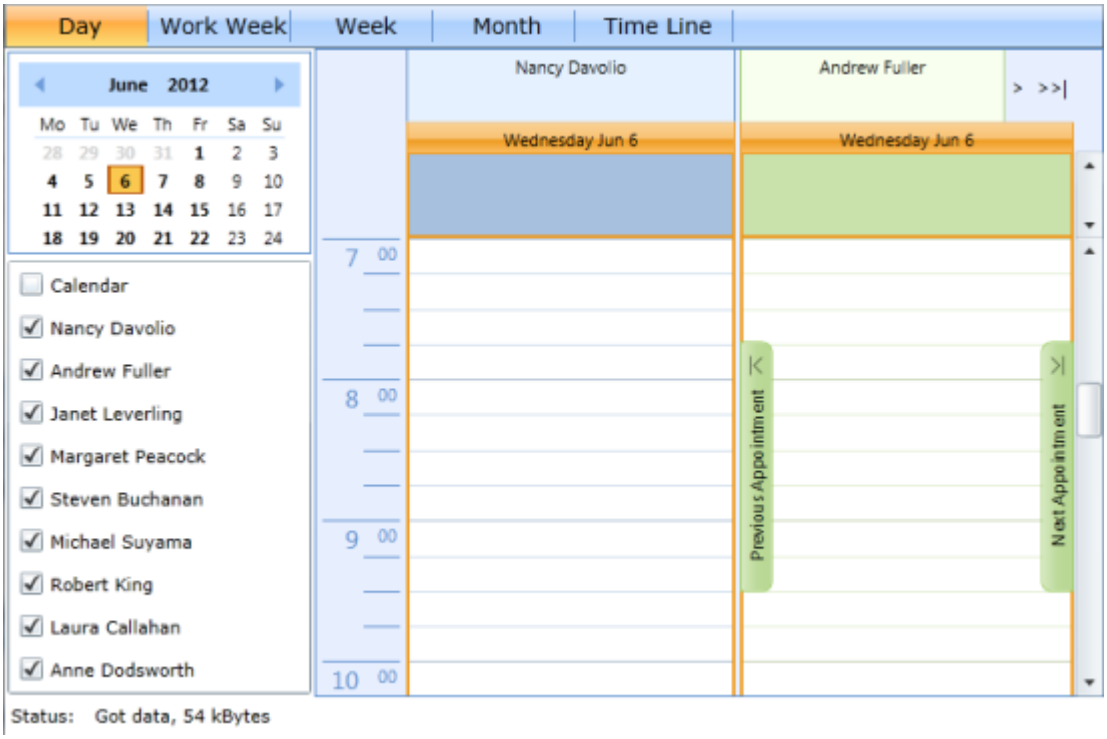
</ResourceDictionary>
```

In this step you created a Resource Dictionary for you application. In the next step, you will complete the application.

Step 4 of 4: Completing the Application

In this step, you will run your application.

Press **F5** to run your application. It should resemble the following image



Note that you can choose how many contacts are displayed on each page. You can also choose the date you wish to display through the C1Calendar controls at the left side of the page. You can also page through the contacts using the arrows at the top of the page.

You created a C1Scheduler control and a custom Schedule view. You also created a custom Resource Dictionary for your application.

Creating a Multi-User Schedule

In this tutorial, you will create an application that will allow you to display schedules for multiple users. You will use XAML markup and code to create the interface. You will also add a Data Service and two code files as resources for your application.

Step 1 of 4: Creating the Application

In this step, you will create a new WPF application, add the appropriate references and namespaces, and add two code files and a Data Service to the application.

Follow these steps:

1. Create a new WPF application in Visual Studio and name it **MultiUser**.
2. Add the following references to your application by right-clicking the References folder in the Solution Explorer and selecting **Add Reference** from the list. Browse to the folder where your references are located and select the following assemblies:
    - C1.WPF
    - C1.WPF.DateTimeEditors
    - C1.WPF.Schedule

3. Right-click the **MultiUser.Web** project and select **Add | Existing Item** from the list. Add the **SmartData** file that applies to your application (either the **SmartData.cs** or the **SmartData.vb** file) and the **PlatformUriTranslator** file that applies to your application (either the **PlatformUriTranslator.cs** or the **PlatformUriTranslator.vb** file).
4. Right-click your application name and select **Add Existing Item** from the list. Locate and select the **Nwind.mdb** database and click **Add**. This will open the **DataSource Configuration Wizard**. Click **Next**.
5. Select **Table** from the list and then click **Finish**. This will add the database and the **NwindDataSet.xsd** files to your application.
6. Add the following namespaces to your **MainWindow.xaml** file:
    - xmlns:local="clr-namespace:MultiUser"
    - xmlns:c1="clr-namespace:C1.WPF;assembly=C1.WPF.Schedule"
    - xmlns:c1sched="clr-namespace:C1.WPF.Schedule;assembly=C1.WPF.Schedule

In this step you created a new application, added the appropriate assembly references, added a database, and added the appropriate namespaces for your application. In the next step you will add XAML to your **MultiUser** application.

Step 2 of 4: Creating the Resources and the C1Scheduler Control

In this step you will create your application resources and data bindings. You will also add and customize a C1Scheduler control.

Follow these steps:

1. Add <*Window.Resources*> <*/Window.Resources*> beneath the namespace declarations.
2. Click between the <*Window.Resources*> tags and add a set of <*ResourceDictionary*> <*/ResourceDictionary*> tags.
3. Click between the <*ResourceDictionary*> tags and add a Merged Dictionary:

XAML

```
<ResourceDictionary.MergedDictionaries>
  <ResourceDictionary
Source="/C1.WPF.Schedule;component/themes/SchedulerThemes/Office2007/Default.xaml"
/>
</ResourceDictionary.MergedDictionaries>
```

4. Directly after the Merged Dictionary, add a set of <*DataTemplate*> <*/DataTemplate*> tags.
5. Click in the opening <*DataTemplate*> tag and add the following XAML markup:

XAML

```
x:Key="myCustomGroupHeaderTemplate"
The opening tag should resemble the following:
<DataTemplate x:Key="myCustomGroupHeaderTemplate">
```

6. Insert the following markup between the <*DataTemplate*> tags to add <*DataTemplate.Resources*>:

XAML

```
<DataTemplate.Resources>
                    <ControlTemplate x:Key="looklessButton" TargetType="{x:Type
```

```
Button}">
                        <Border>
                            <ContentPresenter Margin="4,0"
VerticalAlignment="Center"/>
                        </Border>
                    </ControlTemplate>
                </DataTemplate.Resources>
```

7. Beneath the *<DataTemplate.Resources>* markup, insert a set of *<Grid>* *</Grid>* tags. Click in the first *<Grid>* tag and add SnapsToDevicePixels="True" to the tag.
8. Use the following markup to add row and column definitions to the Grid component:

XAML

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

9. Create the **C1BrushBuilders** and the **Border** control for the **Grid** component:

XAML

```
<c1:C1BrushBuilder x:Name="Background" Style="{StaticResource
C1Scheduler_ControlArea_BrushStyle}" Input="{Binding Background}"/>
                    <c1:C1BrushBuilder x:Name="BorderBrush" Input="{Binding
Background}" Style="{StaticResource
C1Scheduler_MonthHeaderForeground_BrushStyle}"/>
                    <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}"/>
```

10. Add the **ButtonTemplates** and the **TextBlocks** to control the navigation and display for the Scheduler view:

XAML

```
<Button Template="{StaticResource looklessButton}" Content="|&lt;&lt;"
Grid.Column="0" Grid.RowSpan="2" VerticalAlignment="Center"
                                                    FontSize="12"

Command="c1sched:C1Scheduler.NavigateToPreviousGroupCommand"
```

```
CommandParameter="Home" CommandTarget="{Binding Scheduler}"
                                                Visibility="{Binding
ShowPreviousButton, Converter={x:Static
c1sched:BooleanToVisibilityConverter.Default}}"/>
                    <!-- navigate to the previous group -->
                    <Button Template="{StaticResource looklessButton}"
Content="&lt;" Grid.Column="1" Grid.RowSpan="2" VerticalAlignment="Center"
                                                FontSize="12"

Command="c1sched:C1Scheduler.NavigateToPreviousGroupCommand"
CommandTarget="{Binding Scheduler}"
                                                Visibility="{Binding
ShowPreviousButton, Converter={x:Static
c1sched:BooleanToVisibilityConverter.Default}}"/>
                    <!-- navigate to the next group -->
                    <Button Template="{StaticResource looklessButton}"
Content="&gt;" Grid.Column="3" Grid.RowSpan="2" VerticalAlignment="Center"
                                                FontSize="12"

Command="c1sched:C1Scheduler.NavigateToNextGroupCommand" CommandTarget="{Binding
Scheduler}"
                                                Visibility="{Binding
ShowNextButton, Converter={x:Static
c1sched:BooleanToVisibilityConverter.Default}}"/>
                    <!-- navigate to the last group -->
                    <Button Template="{StaticResource looklessButton}"
Content="&gt;&gt;|" Grid.Column="4" Grid.RowSpan="2" VerticalAlignment="Center"
                                                FontSize="12"

Command="c1sched:C1Scheduler.NavigateToNextGroupCommand" CommandParameter="End"
CommandTarget="{Binding Scheduler}"
                                                Visibility="{Binding
ShowNextButton, Converter={x:Static
c1sched:BooleanToVisibilityConverter.Default}}"/>
                    <TextBlock Foreground="{Binding Path=Scheduler.Foreground}"
Margin="10,3" Grid.Column="2"
                                            Visibility="{Binding IsSelected,
Converter={x:Static c1sched: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
c1sched: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"/>
```

11. Create another <*DataTemplate*> to control the group header for the **TimeLine** style:

XAML

```
<DataTemplate x:Key="myCustomTimeLineGroupHeaderTemplate">
                <Grid IsHitTestVisible="False">
                    <c1:C1BrushBuilder x:Name="Background" Style="{StaticResource
C1Scheduler_ControlArea_BrushStyle}" Input="{Binding Background}"/>
                    <c1:C1BrushBuilder x:Name="BorderBrush" Style="{StaticResource
C1Scheduler_MonthHeaderForeground_BrushStyle}" 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>
```

12. Beneath the closing <*/Window.Resources*> tag, insert a set of <*Grid*> <*/Grid*> tags.
13. Insert the following markup between the <*Grid*> tags:

XAML

```
<Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="196"/>
            <ColumnDefinition MinWidth="200"/>
        </Grid.ColumnDefinitions>
```

14. Use the following code within the <*Grid*> tags to create the ToolBar for your application:

XAML

```
<ToolBar Grid.Row="0" Grid.ColumnSpan="2">
            <RadioButton x:Name="btnDay" Content="Day"
                                 CommandTarget="{Binding ElementName=Scheduler}"
                                 Command="c1sched:C1Scheduler.ChangeStyleCommand"
                                 CommandParameter="{Binding Path=OneDayStyle,
ElementName=Scheduler}"/>
            <RadioButton x:Name="btnWorkWeek" Content="Work Week"
                                 CommandTarget="{Binding ElementName=Scheduler}"
                                 Command="c1sched:C1Scheduler.ChangeStyleCommand"
                                 CommandParameter="{Binding Path=WorkingWeekStyle,
ElementName=Scheduler}"/>
            <RadioButton x:Name="btnWeek" Content="Week"
                                 CommandTarget="{Binding ElementName=Scheduler}"
                                 Command="c1sched:C1Scheduler.ChangeStyleCommand"
                                 CommandParameter="{Binding Path=WeekStyle,
ElementName=Scheduler}"/>
            <RadioButton x:Name="btnMonth" Content="Month"
                                 CommandTarget="{Binding ElementName=Scheduler}"
                                 Command="c1sched:C1Scheduler.ChangeStyleCommand"
                                 CommandParameter="{Binding Path=MonthStyle,
ElementName=Scheduler}"/>
            <RadioButton x:Name="btnTimeLine" Content="Time Line"
                                          CommandTarget="{Binding
ElementName=Scheduler}"

Command="c1sched:C1Scheduler.ChangeStyleCommand"
                                          CommandParameter="{Binding
Path=TimeLineStyle, ElementName=Scheduler}"/>
        </ToolBar>
```

15. Finally, create the C1Calendar and C1Scheduler controls and their bindings for your application:

XAML

```
<c1sched:C1Calendar x:Name="Calendar" VerticalAlignment="Stretch" Grid.Column="0"
Grid.Row="1"
                                MaxSelectionCount="42" SelectedDates="{Binding
VisibleDates, ElementName=Scheduler}"
                                CalendarHelper="{Binding CalendarHelper,
ElementName=Scheduler}"
                                BoldedDates="{Binding BoldedDates,
ElementName=Scheduler}"
                                GenerateAdjacentMonthDays="true" Margin="2"/>
        <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>
        <c1sched: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=c1sched:C1Scheduler}}">
            <c1sched:C1Scheduler.Settings>
                <c1sched:C1SchedulerSettings FirstVisibleTime="07:00:00"
AllowContactsEditing="False" AllowCategoriesEditing="False"
AllowCategoriesMultiSelection="False" />
            </c1sched:C1Scheduler.Settings>
        </c1sched:C1Scheduler>
```

In this step you created the application databindings and resources. You also created the C1Scheduler and C1Calendar controls and customized some of the scheduler settings. In the next step, you will add the code to control your application.

Step 3 of 4: Adding the Code to your Application

In this step you will add the code for your application.

1. Right-click your application and select **View Code** from the list.
2. Import the following namespaces:

Visual Basic

```
Imports C1.WPF.Schedule
Imports C1.C1Schedule
Imports System.ComponentModel
Imports System.Collections.Specialized
```

C#

```
using C1.WPF.Schedule;
using System.Collections.Specialized;
using C1.C1Schedule;
```

3. Above the public MainWindow() method, insert the following region:

Visual Basic

```
#Region "** fields"
 Private appointmentsTableAdapter As
MultiUser.NwindDataSetTableAdapters.AppointmentsTableAdapter = New
NwindDataSetTableAdapters.AppointmentsTableAdapter()
 Private employeesTableAdapter As
MultiUser.NwindDataSetTableAdapters.EmployeesTableAdapter = New
NwindDataSetTableAdapters.EmployeesTableAdapter()
 Private customersTableAdapter As
MultiUser.NwindDataSetTableAdapters.CustomersTableAdapter = New
NwindDataSetTableAdapters.CustomersTableAdapter()
 Private dataSet As New NwindDataSet()
#End Region
```

C#

```
#region ** fields
 private MultiUserCS2.NwindDataSetTableAdapters.AppointmentsTableAdapter
appointmentsTableAdapter = new
NwindDataSetTableAdapters.AppointmentsTableAdapter();
 private MultiUserCS2.NwindDataSetTableAdapters.EmployeesTableAdapter
employeesTableAdapter = new NwindDataSetTableAdapters.EmployeesTableAdapter();
 private MultiUserCS2.NwindDataSetTableAdapters.CustomersTableAdapter
customersTableAdapter = new NwindDataSetTableAdapters.CustomersTableAdapter();
 private NwindDataSet dataSet = new NwindDataSet();
#endregion
```

4. Directly below the InitializeComponent() method, insert the following handlers and call to get data from the database:

Visual Basic

```
AddHandler Scheduler.ReminderFire, AddressOf scheduler_ReminderFire
               AddHandler Scheduler.GroupItems.CollectionChanged, AddressOf
GroupItems_CollectionChanged
               ' get data from the data base
               Me.employeesTableAdapter.Fill(dataSet.Employees)
               Me.customersTableAdapter.Fill(dataSet.Customers)
               Me.appointmentsTableAdapter.Fill(dataSet.Appointments)
```

C#

```
Scheduler.ReminderFire += new EventHandler<ReminderActionEventArgs>
(scheduler_ReminderFire);
            Scheduler.GroupItems.CollectionChanged += new
NotifyCollectionChangedEventHandler(GroupItems_CollectionChanged);
            // get data from the data base
```

```
            this.employeesTableAdapter.Fill(dataSet.Employees);
            this.customersTableAdapter.Fill(dataSet.Customers);
           this.appointmentsTableAdapter.Fill(dataSet.Appointments);
```

5. Set the mappings and DataSource for the AppointmentStorage:

Visual Basic

```
Dim storage As AppointmentStorage = 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
```

C#

```
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;
```

6. Set the mappings and DataSOurce for the OwnerStorage:

Visual Basic

```
Dim ownerStorage As ContactStorage = Scheduler.DataStorage.OwnerStorage
               AddHandler(CType(ownerStorage.Contacts,
INotifyCollectionChanged)).CollectionChanged, AddressOf Owners_CollectionChanged
               ownerStorage.Mappings.IndexMapping.MappingName = "EmployeeId"
               ownerStorage.Mappings.TextMapping.MappingName = "FirstName"
               ownerStorage.DataSource = dataSet.Employees
```

C#

```
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;
```

7. Set the mappings and DataSource for the ContactStorage:

Visual Basic

```
Dim cntStorage As ContactStorage = Scheduler.DataStorage.ContactStorage
                AddHandler(CType(cntStorage.Contacts,
INotifyCollectionChanged)).CollectionChanged, AddressOf
Contacts_CollectionChanged
                cntStorage.Mappings.IdMapping.MappingName = "CustomerId"
                cntStorage.Mappings.TextMapping.MappingName = "CompanyName"
                cntStorage.DataSource = dataSet.Customers
                btnDay.IsChecked = True
                AddHandler Scheduler.StyleChanged, AddressOf
Scheduler_StyleChanged
```

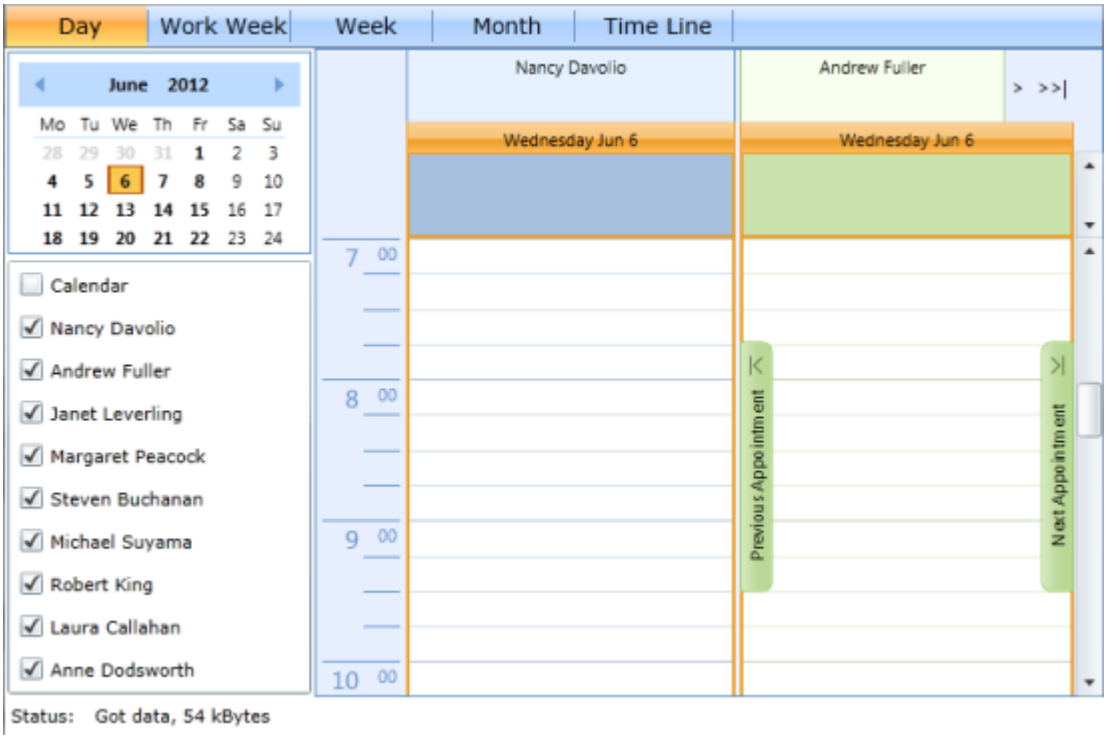
C#

```
// set mappings and DataSource for the ContactStorage
            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;
            btnDay.IsChecked = true;
            Scheduler.StyleChanged += new System.EventHandler<RoutedEventArgs>
(Scheduler_StyleChanged);
```

Step 4 of 4: Running the Application

In this step you will run the application.

Press **F5** to run your application. The application should appear similar to the following image:



Note that you can control the users displayed in the listbox at the left. You can also add an appointment to any user's schedule by double-clicking on the day. You can also control the view with the buttons at the top of the application.

What You've Accomplished

You created a new Silverlight application and added code files and a Data Service. You also added an Access database as a resource for your application. You used code and XAML markup to create and control the design and function of your application.

Scheduler for Silverlight Tutorials

The following tutorials assume that you are familiar with programming in Visual Basic .NET or C#. The tutorials provide step-by-step instructions; no prior knowledge of C1Scheduler is needed. By following the steps outlined in this section, you will be able to create projects demonstrating C1Scheduler features.

You are encouraged to run the tutorial projects, and experiment with your own modifications

Creating a Custom Application for Custom Data

C1Scheduler allows you to customize your data model for those situations in which you need to associate more data fields to each appointment. This also allows you to attach custom Business Objects to each appointment. You can use the **CustomData** property if you need a few additional data fields that can be saved as text. The **Tag** property allows you to associate a business object with each appointment but not serialize it with each appointment. In this tutorial, you will create an application that consists of three parts: the **MainPage.xaml** file, a custom **EditAppointmentDialog.xaml** file that contains the custom appointments dialog, and a **BusinessObjects.cs** or **BusinessObjects.vb** file that defines your business objects classes.

Step 1 of 5: Creating the Application

In this step you will create the main Scheduler application using the Design View, XAML markup, and Code.

Create a new Silverlight application and add the following references by right-clicking **References** in the Solution Explorer and selecting **Add Reference** from the list:

- C1.Silverlight.dll
- C1.Silverlight.Schedule.dll
- C1.Silverlight.DateTimeEditors.dll

Step 2 of 5: Defining Custom Data Structure

In this step, you will define business object classes (BusinessObjects.cs or BusinessObjects.vb files).

1. Right-click the application name and select **Add | New Item**. Choose **Code File** from the list of installed templates. Name the new code file **BusinessObject.cs** or **BusinessObjects.vb** depending on the code language you are using.
2. Import the following namespaces into the page:

Visual Basic

```
Imports System
Imports System.ComponentModel
Imports System.Collections.ObjectModel
```

C#

```
using System;
using System.ComponentModel;
```

```
using System.Collections.ObjectModel;
```

3. Add the following code to implement the business object class:

Visual Basic

```
' Business object should implement INotifyPropertyChanged interface.
Public Class AppointmentBORow
    Implements INotifyPropertyChanged

    Private _subject As String = ""
    Private _body As String = ""
    Private _location As String = ""
    Private _properties As String = ""
    Private _start As DateTime = DateTime.Today
    Private _end As DateTime = (DateTime.Today + TimeSpan.FromDays(1))
    Private _id As Guid = Guid.NewGuid
    Private _isDeleted As Boolean = False
    'Custom Data
    Private _BOProperty1 As String = ""
    Private _BOProperty2 As String = ""
    Public Sub New()
        MyBase.New()
    End Sub
    Public Property Subject As String
        Get
            Return _subject
        End Get
        Set(value As String)
            If (_subject <> value) Then
                _subject = value
                OnPropertyChanged("Subject")
            End If
        End Set
    End Property
    Public Property BOProperty1 As String
        Get
            Return _BOProperty1
        End Get
        Set(value As String)
            If (_BOProperty1 <> value) Then
                _BOProperty1 = value
                OnPropertyChanged("BOProperty1")
            End If
        End Set
    End Property
    Public Property BOProperty2 As String
        Get
            Return _BOProperty2
        End Get
        Set(value As String)
            If (_BOProperty2 <> value) Then
```

```
            _BOProperty2 = value
            OnPropertyChanged("BOProperty2")
        End If
    End Set
End Property
Public Property Properties As String
    Get
        Return _properties
    End Get
    Set(value As String)
        If (_properties <> value) Then
            _properties = value
            OnPropertyChanged("Properties")
        End If
    End Set
End Property
Public Property Start As DateTime
    Get
        Return _start
    End Get
    Set(value As DateTime)
        If (_start <> value) Then
            _start = value
            OnPropertyChanged("Start")
        End If
    End Set
End Property
Public Property dt As DateTime
    Get
        Return _end
    End Get
    Set(value As DateTime)
        If (_end <> value) Then
            _end = value
            OnPropertyChanged("End")
        End If
    End Set
End Property
Public Property Body As String
    Get
        Return _body
    End Get
    Set(value As String)
        If (_body <> value) Then
            _body = value
            OnPropertyChanged("Body")
        End If
    End Set
End Property
Public Property Location As String
    Get
```

```
            Return _location
        End Get
        Set(value As String)
            If (_location <> value) Then
                _location = value
                OnPropertyChanged("Location")
            End If
        End Set
    End Property
    Public Property Id As Guid
        Get
            Return _id
        End Get
        Set(value As Guid)
            If (_id <> value) Then
                _id = value
                OnPropertyChanged("Id")
            End If
        End Set
    End Property
    Public Property IsDeleted As Boolean
        Get
            Return _isDeleted
        End Get
        Set(value As Boolean)
            If (_isDeleted <> value) Then
                _isDeleted = value
                OnPropertyChanged("IsDeleted")
            End If
        End Set
    End Property

    Public Event PropertyChanged As PropertyChangedEventHandler
    Protected Overridable Sub OnPropertyChanged(ByVal propName As String)
        If PropertyChangedEvent IsNot Nothing Then
            RaiseEvent PropertyChanged(Me, New
PropertyChangedEventArgs(propName))
        End If
    End Sub

    Public Event PropertyChanged1(sender As Object, e As
System.ComponentModel.PropertyChangedEventArgs) Implements
System.ComponentModel.INotifyPropertyChanged.PropertyChanged
End Class
```

C#

```
// Business object should implement INotifyPropertyChanged interface.
    public class AppointmentBORow : INotifyPropertyChanged
    {
        private string _subject = "";
        private string _body = "";
```

```
        private string _location = "";
        private string _properties = "";
        private DateTime _start = DateTime.Today;
        private DateTime _end = DateTime.Today + TimeSpan.FromDays(1);
        private Guid _id = Guid.NewGuid();
        private bool _isDeleted = false;
        //Custom Data
        private string _BOProperty1 = "";
        private string _BOProperty2 = "";
        public AppointmentBORow()
        {
        }
        public string Subject
        {
            get { return _subject; }
            set
            {
                if (_subject != value)
                {
                    _subject = value;
                    OnPropertyChanged("Subject");
                }
            }
        }
        public string BOProperty1
        {
            get { return _BOProperty1; }
            set
            {
                if (_BOProperty1 != value)
                {
                    _BOProperty1 = value;
                    OnPropertyChanged("BOProperty1");
                }
            }
        }
        public string BOProperty2
        {
            get { return _BOProperty2; }
            set
            {
                if (_BOProperty2 != value)
                {
                    _BOProperty2 = value;
                    OnPropertyChanged("BOProperty2");
                }
            }
        }
        public string Properties
        {
            get { return _properties; }
```

```
        set
        {
            if (_properties != value)
            {
                _properties = value;
                OnPropertyChanged("Properties");
            }
        }
    }
    public DateTime Start
    {
        get { return _start; }
        set
        {
            if (_start != value)
            {
                _start = value;
                OnPropertyChanged("Start");
            }
        }
    }
    public DateTime End
    {
        get { return _end; }
        set
        {
            if (_end != value)
            {
                _end = value;
                OnPropertyChanged("End");
            }
        }
    }
    public string Body
    {
        get { return _body; }
        set
        {
            if (_body != value)
            {
                _body = value;
                OnPropertyChanged("Body");
            }
        }
    }
    public string Location
    {
        get { return _location; }
        set
        {
            if (_location != value)
```

```
                {
                    _location = value;
                    OnPropertyChanged("Location");
                }
            }
        }
        public Guid Id
        {
            get { return _id; }
            set
            {
                if (_id != value)
                {
                    _id = value;
                    OnPropertyChanged("Id");
                }
            }
        }
        public bool IsDeleted
        {
            get { return _isDeleted; }
            set
            {
                if (_isDeleted != value)
                {
                    _isDeleted = value;
                    OnPropertyChanged("IsDeleted");
                }
            }
        }
        public event PropertyChangedEventHandler PropertyChanged;
        protected virtual void OnPropertyChanged(string propName)
        {
            if (PropertyChanged != null)
                PropertyChanged(this, new PropertyChangedEventArgs(propName));
        }
    }
```

4. Insert the following code to implement the business object collection class:

Visual Basic

```
' Business objects collection should implement INotifyCollectionChanged
interface.
Public Class AppointmentBOList
    Inherits ObservableCollection(Of AppointmentBORow)
    Protected Overrides Sub RemoveItem(ByVal index As Integer)
        Dim item As AppointmentBORow = Me(index)
        If (Not (item) Is Nothing) Then
            item.IsDeleted = True
        End If
        MyBase.RemoveItem(index)
```

```
    End Sub
End Class
```

C#

```
// Business objects collection should implement INotifyCollectionChanged
interface.
    public class AppointmentBOList : ObservableCollection<AppointmentBORow>
    {
        protected override void RemoveItem(int index)
        {
            AppointmentBORow item = this[index];
            if (item != null)
            {
                item.IsDeleted = true;
            }
            base.RemoveItem(index);
        }
    }
}
```

5. Insert the following public class below the namespace declaration (it creates and initializes the instance of the business objects collection):

Visual Basic

```
Public Class AppointmentsBO
    Private _list As AppointmentBOList = Nothing
    Public Sub New()
        MyBase.New()
        _list = New AppointmentBOList
        _list.Add(New AppointmentBORow)
    End Sub
    Public Property Appointments As AppointmentBOList
        Get
            Return _list
        End Get
        Set(value As AppointmentBOList)
            _list = value
        End Set
    End Property
End Class
```

C#

```
public class AppointmentsBO
    {
        AppointmentBOList _list = null;

        public AppointmentsBO()
        {
            _list = new AppointmentBOList();
            _list.Add(new AppointmentBORow());
```

```
        }

        public AppointmentBOList Appointments
        {
            get
            {
                return _list;
            }
            set
            {
                _list = value;
            }
        }
    }
```

In this step you created business objects which will provide data for the C1Scheduler control in the future steps.

Step 3 of 5: Creating the Custom Appointment Dialog

In this step, you will create the **Custom Appointment Dialog** and set the code to control the dialog using XAML markup and code.

1. Create a new Silverlight page by right-clicking the application name and selecting **Add | New Item** from the list.
2. Select **New Silverlight Page** from the list of installed templates and name it **EditAppointmentDialog**. Click **OK**.
3. Replace the *<navigation:Page> </navigation:Page>* tags and the namespaces with the following *<UserControl>* statement:

XAML

```
<UserControl x:Class="SchedulerCustomData.EditAppointmentDialog"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:c1sched="clr-namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule"
xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
xmlns:local="clr-namespace:SchedulerCustomData"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="400">
```

4. Insert the following XAML after the namespace declarations to create the custom **Appointment** dialog box:

XAML

```
<Grid x:Name="LayoutRoot" Background="White"
BindingValidationError="LayoutRoot_BindingValidationError">
        <Grid.Resources>
<res:C1_Schedule_EditAppointment x:Key="C1_Schedule_EditAppointment"/>
        </Grid.Resources>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
```

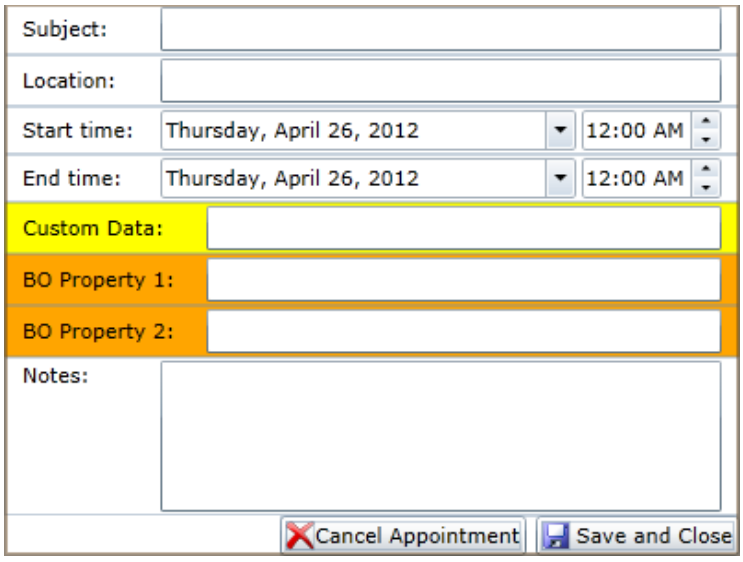
```
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition  />
            <RowDefinition Height="Auto" />
        </Grid.RowDefinitions>
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="75" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
<TextBlock VerticalAlignment="Center" Margin="10,2,0,2" HorizontalAlignment="Left"
Text="{Binding subject, Source={StaticResource C1_Schedule_EditAppointment}}" />
<TextBox x:Name="subject" Grid.Column="1" TabIndex="0" Padding="2" Text="{Binding
Subject, Mode=TwoWay}" TextChanged="subject_TextChanged" MaxLength="255"
VerticalAlignment="Center" Margin="10,2,10,2" />
        </Grid>
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="75" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
        <TextBlock VerticalAlignment="Center" Margin="10,2,0,2"
HorizontalAlignment="Left" Text="{Binding location, Source={StaticResource
C1_Schedule_EditAppointment}}" />
<TextBox TabIndex="1" x:Name="location" Grid.Column="1" Text="{Binding Location,
Mode=TwoWay}" MaxLength="255" VerticalAlignment="Center" Margin="10,2,10,2" />
        </Grid>
        <Grid Grid.Row="2">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="75" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
 <TextBlock VerticalAlignment="Center" Margin="10,2,0,2" HorizontalAlignment="Left"
Text="{Binding startTime, Source={StaticResource C1_Schedule_EditAppointment}}"/>
<c1datetime:C1DateTimePicker x:Name="startCalendar" Grid.Column="1"
VerticalAlignment="Center" Margin="10,2,10,2"        Padding="1" IsTabStop="False"
TimeFormat="ShortTime" DateFormat="Long" FirstDayOfWeek="{Binding
Path=ParentCollection.ParentStorage.ScheduleStorage.Scheduler.CalendarHelper.WeekStart}"
DateTime="{Binding Start, Mode=TwoWay}"/>
        </Grid>
        <Grid Grid.Row="3">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="75" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
 <TextBlock VerticalAlignment="Center" Margin="10,2,0,2" HorizontalAlignment="Left"
Text="{Binding endTime, Source={StaticResource C1_Schedule_EditAppointment}}"/>
<c1datetime:C1DateTimePicker x:Name="endCalendar" VerticalAlignment="Center"
Margin="10,2,10,2" Padding="1" DateTimeChanged="endCalendar_DateTimeChanged"
FirstDayOfWeek="{Binding
Path=ParentCollection.ParentStorage.ScheduleStorage.Scheduler.CalendarHelper.WeekStart}"
TimeFormat="ShortTime" DateFormat="Long" Grid.Column="1" c1:C1NagScreen.Nag="True" />
        </Grid>
        <Grid Grid.Row="4" Background="Yellow">
            <Grid.ColumnDefinitions>
```

```
                <ColumnDefinition Width="100" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
<TextBlock VerticalAlignment="Center" Text="Custom Data:" HorizontalAlignment="Left"
Margin="10,2,0,2"/>
<TextBox Name="customDataTextBox" Grid.Column="1" Text="{Binding Path=CustomData,
Mode=TwoWay}" Margin="10,2,10,2"/>
        </Grid>
        <Grid Grid.Row="5" Background="Orange">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="100" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
<TextBlock VerticalAlignment="Center" Text="BO Property 1:" HorizontalAlignment="Left"
Margin="10,2,0,2"/>
<TextBox Name="BOTextBox1" Grid.Column="1" Text="{Binding Path=Tag.Tag.BOProperty1,
Mode=TwoWay}" Margin="10,2,10,2" />
        </Grid>
        <Grid Grid.Row="6" Background="Orange">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="100" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>

<TextBlock VerticalAlignment="Center" Text="BO Property 2:" HorizontalAlignment="Left"
Margin="10,2,0,2"/>
<TextBox Name="BOTextBox2" Grid.Column="1" Text="{Binding Path=Tag.Tag.BOProperty2,
Mode=TwoWay}" Margin="10,2,10,2"/>
        </Grid>
        <Grid Grid.Row="7">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="75" />
                <ColumnDefinition Width="*" />
            </Grid.ColumnDefinitions>
<TextBlock VerticalAlignment="Top" Text="Notes:" HorizontalAlignment="Left"
Margin="10,2,0,2"/>
<TextBox x:Name="body" Text="{Binding Body, Mode=TwoWay}" AcceptsReturn="true"
Grid.Column="1" Margin="10,2,10,2" />
        </Grid>
<StackPanel Orientation="Horizontal" Grid.Row="8" HorizontalAlignment="Right">
<Button Name="deleteButton" IsTabStop="False"  Margin="0,0,5,0"
c1:CommandExtensions.CommandTarget="{Binding
ParentCollection.ParentStorage.ScheduleStorage.Scheduler}"
c1:CommandExtensions.CommandParameter="{Binding Tag}"
c1:CommandExtensions.Command="c1sched:C1Scheduler.DeleteAppointmentCommand"
ToolTipService.ToolTip="{Binding deleteButton_ToolTip, Source={StaticResource
C1_Schedule_EditAppointment}}">
<StackPanel Orientation="Horizontal" VerticalAlignment="Center">
<Image Height="16" Width="16" Source="/SchedulerCustomData;component/Images/Delete.png">
</Image>
  <TextBlock Text="Cancel Appointment" />
                </StackPanel>
            </Button>
 <Button x:Name="PART_DialogSaveButton" IsTabStop="False"
Click="PART_DialogSaveButton_Click"       ToolTipService.ToolTip="{Binding
PART_DialogSaveButton_Tooltip, Source={StaticResource C1_Schedule_EditAppointment}}">
```

```
<StackPanel Orientation="Horizontal" VerticalAlignment="Center">
 <Image Height="16" Width="16" Source="/SchedulerCustomData;component/Images/save.png">
</Image>
<TextBlock Margin="3,0,0,0" Text="{Binding PART_DialogSaveButton_AccessText, Source=
{StaticResource C1_Schedule_EditAppointment}}" VerticalAlignment="Center"/>
</StackPanel>
</Button>
</StackPanel>
    </Grid>
</UserControl>
```

The **Edit Appointment** dialog box in Design View should resemble the following image:



5. Right-click on the page and select **View Code** from the list. In Code View, add the following namespaces to the application:

Visual Basic

```
Imports C1.Silverlight.Schedule
Imports C1.Silverlight
Imports C1.Silverlight.DateTimeEditors
Imports C1.C1Schedule
Imports System.Windows.Data
```

C#

```
using C1.Silverlight.Schedule;
using C1.Silverlight;
using C1.Silverlight.DateTimeEditors;
using C1.C1Schedule;
using System.Windows.Data;
```

6. Add the following region directive and public method below the public partial class:

VIsual Basic

```
#Region "fields"
    Dim _parentWindow As ContentControl = Nothing
```

```
    Dim _appointment As Appointment
    Dim _scheduler As C1Scheduler
#End Region
```

C#

```
#region ** fields
        private ContentControl _parentWindow = null;
        private Appointment _appointment;
        //AppointmentBORow customBO; //not used
        private C1Scheduler _scheduler;
        #endregion
```

7. Add a **UserControl_Loaded** event handler:

Visual Basic

```
Private Sub UserControl_Loaded(ByVal sender As Object, ByVal e As RoutedEventArgs)
        _appointment = CType(DataContext, Appointment)
        'Appointment sourceApp = _appointment.Tag As Appointment
        'customBO = sourceApp.Tag As AppointmentBORow
        _parentWindow = CType(VTreeHelper.GetParentOfType(Me, GetType(C1Window)),
ContentControl)
        If (Not (_parentWindow) Is Nothing) Then
            Dim bnd As Binding = New Binding("Header")
            bnd.Source = Me
            _parentWindow.SetBinding(C1Window.HeaderProperty, bnd)
        End If
        If (Not (_appointment) Is Nothing) Then
            AddHandler CType(_appointment,
System.ComponentModel.INotifyPropertyChanged).PropertyChanged, AddressOf
Me.appointment_PropertyChanged
            If (Not (_appointment.ParentCollection) Is Nothing) Then
                _scheduler =
_appointment.ParentCollection.ParentStorage.ScheduleStorage.Scheduler
            End If
            Header = "Edit Appointment"
            UpdateEndCalendar()
            endCalendar.EditMode = C1DateTimePickerEditMode.DateTime
            startCalendar.EditMode = C1DateTimePickerEditMode.DateTime
        End If
        subject.Focus()
    End Sub
```

C#

```
 private void UserControl_Loaded(object sender, RoutedEventArgs e)
        {
            _appointment = DataContext as Appointment;
            //Appointment sourceApp = _appointment.Tag as Appointment;
            //customBO = sourceApp.Tag as AppointmentBORow;

            _parentWindow = (ContentControl)VTreeHelper.GetParentOfType(this,
typeof(C1Window));
            if (_parentWindow != null)
            {
```

```
                Binding bnd = new Binding("Header");
                bnd.Source = this;
                _parentWindow.SetBinding(C1Window.HeaderProperty, bnd);
            }
            if (_appointment != null)
            {

((System.ComponentModel.INotifyPropertyChanged)_appointment).PropertyChanged += new
System.ComponentModel.PropertyChangedEventHandler(appointment_PropertyChanged);
                if (_appointment.ParentCollection != null)
                {
                    _scheduler =
_appointment.ParentCollection.ParentStorage.ScheduleStorage.Scheduler;
                }
                Header = "Edit Appointment";
                UpdateEndCalendar();
                startCalendar.EditMode = endCalendar.EditMode =
C1DateTimePickerEditMode.DateTime;
            }
            subject.Focus();
        }
```

8. Add the following code to govern some of the button and mouse events:

Visual Basic

```
Protected Overrides Sub OnMouseEnter(ByVal e As MouseEventArgs)
        MyBase.OnMouseEnter(e)
    End Sub

    Protected Overrides Sub OnGotFocus(ByVal e As RoutedEventArgs)
        MyBase.OnGotFocus(e)
    End Sub

    Private Sub LayoutRoot_BindingValidationError(ByVal sender As Object, ByVal e As
ValidationErrorEventArgs)
        If (e.Action = ValidationErrorEventAction.Added) Then
            PART_DialogSaveButton.IsEnabled = False
            PART_DialogSaveButton.IsEnabled = False
        Else
            PART_DialogSaveButton.IsEnabled = True
        End If
    End Sub

    Private Sub PART_DialogSaveButton_Click(ByVal sender As Object, ByVal e As
RoutedEventArgs)
        SetAppointment()
        CType(_parentWindow, C1Window).DialogResult = MessageBoxResult.OK
    End Sub

    Private Sub subject_TextChanged(ByVal sender As Object, ByVal e As
TextChangedEventArgs)
        subject.GetBindingExpression(TextBox.TextProperty).UpdateSource()
    End Sub
```

C#

```
protected override void OnMouseEnter(MouseEventArgs e)
        {
            base.OnMouseEnter(e);
        }
        protected override void OnGotFocus(RoutedEventArgs e)
        {
            base.OnGotFocus(e);
        }
        private void LayoutRoot_BindingValidationError(object sender,
ValidationErrorEventArgs e)
        {
            if (e.Action == ValidationErrorEventAction.Added)
            {
                PART_DialogSaveButton.IsEnabled = false;
            }
            else
            {
                PART_DialogSaveButton.IsEnabled = true;
            }
        }
        private void PART_DialogSaveButton_Click(object sender, RoutedEventArgs e)
        {
            SetAppointment();
            ((C1Window)_parentWindow).DialogResult = MessageBoxResult.OK;
        }
        private void subject_TextChanged(object sender, TextChangedEventArgs e)
        {           subject.GetBindingExpression(TextBox.TextProperty).UpdateSource();
        }
```

9. Add the **SetAppointment** method :

Visual Basic

```
Private Sub SetAppointment()
        location.GetBindingExpression(TextBox.TextProperty).UpdateSource()
        body.GetBindingExpression(TextBox.TextProperty).UpdateSource()
        'Update Additional Data Properties
        customDataTextBox.GetBindingExpression(TextBox.TextProperty).UpdateSource()
        BOTextBox1.GetBindingExpression(TextBox.TextProperty).UpdateSource()
        BOTextBox2.GetBindingExpression(TextBox.TextProperty).UpdateSource()
    End Sub

    Private Sub appointment_PropertyChanged(ByVal sender As Object, ByVal e As
System.ComponentModel.PropertyChangedEventArgs)
        UpdateEndCalendar()
    End Sub
```

C#

```
private void SetAppointment()
        {            location.GetBindingExpression(TextBox.TextProperty).UpdateSource();
body.GetBindingExpression(TextBox.TextProperty).UpdateSource();
            //Update Additional Data Properties
customDataTextBox.GetBindingExpression(TextBox.TextProperty).UpdateSource();
BOTextBox1.GetBindingExpression(TextBox.TextProperty).UpdateSource();
BOTextBox2.GetBindingExpression(TextBox.TextProperty).UpdateSource();
        }
```

```
        void appointment_PropertyChanged(object sender,
System.ComponentModel.PropertyChangedEventArgs e)
        {
            UpdateEndCalendar();
        }
```

10. Add the following object model to the **EditAppointmentDialog** class:

Visual Basic

```
#Region "Appointment Properties"
    Public Property Appointment As Appointment
        Get
            Return _appointment
        End Get
        Set(value As Appointment)
            _appointment = value
            If (Not (_parentWindow) Is Nothing) Then
                _parentWindow.DataContext = value
                _parentWindow.Content = value
            End If
            DataContext = value
        End Set
    End Property
    ''' <summary>
    ''' Gets a <see cref="String"/> value which can be used as an Appointment window
header.
    ''' </summary>
    Public Property Header As String
        Get
            Return CType(GetValue(HeaderProperty), String)
        End Get
        Set(value As String)
            SetValue(HeaderProperty, value)
        End Set
    End Property
    Private Shared HeaderProperty As DependencyProperty =
DependencyProperty.Register("Header", GetType(System.String),
GetType(EditAppointmentDialog), Nothing)
    ''' <summary>
    ''' Gets recurrence pattern description.
    ''' </summary>
    Public Property PatternDescription As String
        Get
            Return CType(GetValue(PatternDescriptionProperty), String)
        End Get
        Set(value As String)
            SetValue(PatternDescriptionProperty, value)
        End Set
    End Property
    Public Shared PatternDescriptionProperty As DependencyProperty =
DependencyProperty.Register("PatternDescription", GetType(System.String),
GetType(EditAppointmentControl), New PropertyMetadata(String.Empty))
#End Region
```

C#

```
#region ** object model
        /// <summary>
        /// Gets an <see cref="Appointment"/> object representing current DataContext.
        /// </summary>
        public Appointment Appointment
        {
            get
            {
                return _appointment;
            }
            set
            {
                _appointment = value;
                if (_parentWindow != null)
                {
                    _parentWindow.Content =
                    _parentWindow.DataContext = value;
                }
                DataContext = value;
            }
        }
        /// <summary>
        /// Gets a <see cref="String"/> value which can be used as an Appointment window
header.
        /// </summary>
        public string Header
        {
            get { return (string)GetValue(HeaderProperty); }
            private set { SetValue(HeaderProperty, value); }
        }
        private static readonly DependencyProperty HeaderProperty =
            DependencyProperty.Register("Header", typeof(string),
typeof(EditAppointmentDialog), null);
        /// <summary>
        /// Gets recurrence pattern description.
        /// </summary>
        public string PatternDescription
        {
            get { return (string)GetValue(PatternDescriptionProperty); }
            private set { SetValue(PatternDescriptionProperty, value); }
        }
        public static readonly DependencyProperty PatternDescriptionProperty =
            DependencyProperty.Register("PatternDescription", typeof(string),
            typeof(EditAppointmentControl), new PropertyMetadata(string.Empty));
        #endregion
```

11. Insert the last portion of code to control the **DateTimeChanged** method:

Visual Basic

```
#Region "DateTime Picker logic"
    Private Sub endCalendar_DateTimeChanged(ByVal sender As Object, ByVal e As
NullablePropertyChangedEventArgs(Of DateTime))
        If (Not (_appointment) Is Nothing) Then
            Dim dt As DateTime = endCalendar.DateTime.Value
            If _appointment.AllDayEvent Then
                dt = dt.AddDays(1)
```

```
            End If
            If (dt < Appointment.Start) Then
                endCalendar.Background = New SolidColorBrush(Colors.Red)
                endCalendar.Foreground = New SolidColorBrush(Colors.Red)
                endCalendar.BorderBrush = New SolidColorBrush(Colors.Red)
                endCalendar.BorderThickness = New Thickness(2)
                ToolTipService.SetToolTip(endCalendar,
C1.Silverlight.Schedule.Resources.C1_Schedule_Exceptions.StartEndValidationFailed)
                PART_DialogSaveButton.IsEnabled = False
            Else
                _appointment.End = dt
                If Not PART_DialogSaveButton.IsEnabled Then
                    PART_DialogSaveButton.IsEnabled = True
                    endCalendar.ClearValue(Control.BackgroundProperty)
                    endCalendar.ClearValue(Control.ForegroundProperty)
                    endCalendar.ClearValue(Control.BorderBrushProperty)
                    endCalendar.ClearValue(Control.BorderThicknessProperty)
                    endCalendar.ClearValue(ToolTipService.ToolTipProperty)
                End If
            End If
        End If
    End Sub
    Private Sub UpdateEndCalendar()
        Dim dt As DateTime = _appointment.End
        If _appointment.AllDayEvent Then
            dt = dt.AddDays(-1)
        End If
        endCalendar.DateTime = dt
        If Not PART_DialogSaveButton.IsEnabled Then
            PART_DialogSaveButton.IsEnabled = True
            endCalendar.ClearValue(Control.BackgroundProperty)
            endCalendar.ClearValue(Control.ForegroundProperty)
            endCalendar.ClearValue(Control.BorderBrushProperty)
            endCalendar.ClearValue(Control.BorderThicknessProperty)
            endCalendar.ClearValue(ToolTipService.ToolTipProperty)
        End If
    End Sub
#End Region
```

C#

```
#region "DateTime Picker logic"
        private void endCalendar_DateTimeChanged(object sender,
NullablePropertyChangedEventArgs<DateTime> e)
        {
            if (_appointment != null)
            {
                DateTime end = endCalendar.DateTime.Value;
                if (_appointment.AllDayEvent)
                {
                    end = end.AddDays(1);
                }
                if (end < Appointment.Start)
                {
                    endCalendar.BorderBrush = endCalendar.Foreground =
                    endCalendar.Background = new SolidColorBrush(Colors.Red);
```

```
                    endCalendar.BorderThickness = new Thickness(2);
                    ToolTipService.SetToolTip(endCalendar,
C1.Silverlight.Schedule.Resources.C1_Schedule_Exceptions.StartEndValidationFailed);
                    PART_DialogSaveButton.IsEnabled = false;
                }
                else
                {
                    _appointment.End = end;
                    if (!PART_DialogSaveButton.IsEnabled)
                    {
                        PART_DialogSaveButton.IsEnabled = true;
endCalendar.ClearValue(Control.BackgroundProperty);
endCalendar.ClearValue(Control.ForegroundProperty);
endCalendar.ClearValue(Control.BorderBrushProperty);
endCalendar.ClearValue(Control.BorderThicknessProperty);
endCalendar.ClearValue(ToolTipService.ToolTipProperty);
                    }
                }
            }
        }
        private void UpdateEndCalendar()
        {
            DateTime end = _appointment.End;
            if (_appointment.AllDayEvent)
            {
                end = end.AddDays(-1);
            }
            endCalendar.DateTime = end;
            if (!PART_DialogSaveButton.IsEnabled)
            {
                PART_DialogSaveButton.IsEnabled = true;
                endCalendar.ClearValue(Control.BackgroundProperty);
                endCalendar.ClearValue(Control.ForegroundProperty);
                endCalendar.ClearValue(Control.BorderBrushProperty);
endCalendar.ClearValue(Control.BorderThicknessProperty);
endCalendar.ClearValue(ToolTipService.ToolTipProperty);
            }
        }
        #endregion
    }
}
```

In this step, you created the custom **Edit Appointment** dialog box and set the code to control the appointment methods and events.

Step 4 of 5: Adding Functionality to the Application Main Page

In this step, you will add a C1Scheduler control to the main page of your application and set its properties so that it uses your custom appointment dialog to edit appointments and custom business objects collection as data source for the **AppointmentStorage**.

1. Open **MainPage.xaml** file and add the following XAML namespaces to the <UserControl> tag:

XAML
`xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"`

```
xmlns:c1sched="clr-namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule"
xmlns:local=" clr-namespace:SchedulerCustomData"
```

The <UserControl> tag should resemble the following:

XAML

```
<UserControl x:Class="SchedulerCustomData.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc=http://schemas.openxmlformats.org/markup-compatibility/2006
xmlns:local=" clr-namespace:SchedulerCustomData"
xmlns:c1sched="clr-namespace:C1.C1Schedule;assembly=C1.Silverlight.Schedule.5"
Loaded="UserControl_Loaded">
```

2. Define custom DataTemplate for editing appointments in the page resources:

XAML

```
<UserControl.Resources>
        <DataTemplate x:Key="customEditAppointmentTemplate">
            <local:EditAppointmentDialog/>
        </DataTemplate>
    </UserControl.Resources>
```

3. Add a C1Scheduler control to your application by locating the control in the Toolbox and double-clicking it to add it to your application.
4. Add the following XAML to the <C1Schedule> tag set the C1Scheduler properties and event handlers:

XAML

```
Height="300" HorizontalAlignment="Left" Name="c1Scheduler1" VerticalAlignment="Top"
Width="400" Style="{Binding OneDayStyle, RelativeSource={RelativeSource Self}}"
EditAppointmentTemplate="{StaticResource customEditAppointmentTemplate}"
UserAddingAppointment="c1Scheduler1_UserAddingAppointment"
AppointmentAdded="c1Scheduler1_AppointmentAdded"
UserEditingAppointment="c1Scheduler1_UserEditingAppointment"
```

5. Right-click the application and select **View Code** from the list to open **MainPage.xaml.cs** or the **MainPage.xaml.vb**.
6. Add the following namespaces:

Visual Basic

```
Imports C1.Silverlight.Schedule
Imports C1.Silverlight
Imports C1.Silverlight.DateTimeEditors
Imports C1.C1Schedule
Imports System.Windows.Data
```

C#

```
using C1.Silverlight.Schedule;
using C1.Silverlight;
using C1.Silverlight.DateTimeEditors;
using C1.C1Schedule;
```

```
using System.Windows.Data;
```

7. Create the **AppointmentBOList** class instance as a field in the **MainPage** class:

Visual Basic

```
 Dim myAppointments As AppointmentsBO = New AppointmentsBO
```

C#

```
AppointmentsBO myAppointments = new AppointmentsBO();
```

8. Insert the following code into the **MainPage** constructor directly below the **InitializeComponent() method** call to create a sample appointment:

Visual Basic

```
 Dim app1 As AppointmentBORow = New AppointmentBORow
        app1.Body = "Test Body"
        app1.dt = (DateTime.Now + TimeSpan.FromHours(1))
        app1.Start = DateTime.Now
        app1.Location = "Test Location"
        app1.Id = Guid.NewGuid
        app1.Subject = "Test Subject"
```

C#

```
// Create a sample appointment
AppointmentBORow app1 = new AppointmentBORow();
app1.Body = "Test Body";
app1.End = DateTime.Now + TimeSpan.FromHours(1);
app1.Start = DateTime.Now;
app1.Location = "Test Location";
app1.Id = Guid.NewGuid();
app1.Subject = "Test Subject";
```

9. Fill the custom properties for the Business Objects by inserting the following code:

Visual Basic

```
' Custom BO Properties
app1.BOProperty1 = "New Property"
app1.BOProperty2 = "Second Property"
myAppointments.Add(app1)
```

C#

```
// Custom BO Properties
app1.BOProperty1 = "New Property";
app1.BOProperty2 = "Second Property";
myAppointments.Add(app1);
```

10. Next, map AppointmentStorage to custom business object properties and set **AppointmentStorage.DataSource** to the business objects collection instance:

Visual Basic

```
'Data binding
```

```
 c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Body.MappingName = "Body"
        c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Start.MappingName =
"Start"
        c1Scheduler1.DataStorage.AppointmentStorage.Mappings.End.MappingName = "End"
        c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Location.MappingName =
"Location"
        c1Scheduler1.DataStorage.AppointmentStorage.Mappings.IdMapping.MappingName =
"Id"
        c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Subject.MappingName =
"Subject"

c1Scheduler1.DataStorage.AppointmentStorage.Mappings.AppointmentProperties.MappingName
= "Properties"
        c1Scheduler1.DataStorage.AppointmentStorage.DataSource =
myAppointments.Appointments
    End Sub
```

C#

```
//Data binding
c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Body.MappingName = "Body";
c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Start.MappingName = "Start";
c1Scheduler1.DataStorage.AppointmentStorage.Mappings.End.MappingName = "End";
c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Location.MappingName =
"Location";
c1Scheduler1.DataStorage.AppointmentStorage.Mappings.IdMapping.MappingName = "Id";
c1Scheduler1.DataStorage.AppointmentStorage.Mappings.Subject.MappingName = "Subject";
c1Scheduler1.DataStorage.AppointmentStorage.Mappings.AppointmentProperties.MappingName
= "Properties";
c1Scheduler1.DataStorage.AppointmentStorage.DataSource = myAppointments.Appointments;
```

11. Insert the following code to set the **Scheduler Appointment** settings:

Visual Basic

```
Private Sub c1Scheduler1_UserAddingAppointment(ByVal sender As Object, ByVal e As
C1.Silverlight.Schedule.AppointmentActionEventArgs)
        Dim newApp As AppointmentBORow = New AppointmentBORow
        e.Appointment.Tag = newApp
    End Sub
    Private Sub c1Scheduler1_AppointmentAdded(ByVal sender As Object, ByVal e As
C1.Silverlight.Schedule.AppointmentActionEventArgs)
        Dim newApp As AppointmentBORow = FindMyAppointment(CType(e.Appointment.Key(0),
Guid))
        Dim editedApp As AppointmentBORow = CType(e.Appointment.Tag, AppointmentBORow)
        If (Not (editedApp) Is Nothing) Then
            newApp.BOProperty1 = editedApp.BOProperty1
            newApp.BOProperty2 = editedApp.BOProperty2
        End If
    End Sub
    Private Sub c1Scheduler1_UserEditingAppointment(ByVal sender As Object, ByVal e As
C1.Silverlight.Schedule.AppointmentActionEventArgs)
        Dim myApp As AppointmentBORow = FindMyAppointment(CType(e.Appointment.Key(0),
Guid))
        e.Appointment.Tag = myApp
    End Sub
```

```
    Private Function FindMyAppointment(ByVal id As Guid) As AppointmentBORow
        For Each app As AppointmentBORow In myAppointments.Appointments
            If (app.Id = id) Then
                Return app
            End If
        Next
        Return Nothing
    End Function
```

C#

```
//Set schedule interval to 1 hour
            c1Scheduler1.VisualIntervalScale = TimeSpan.FromHours(1);
        }
        private void c1Scheduler1_UserAddingAppointment(object sender,
C1.Silverlight.Schedule.AppointmentActionEventArgs e)
        {
            AppointmentBORow newApp = new AppointmentBORow();
            e.Appointment.Tag = newApp;
        }
        private void c1Scheduler1_AppointmentAdded(object sender,
C1.Silverlight.Schedule.AppointmentActionEventArgs e)
        {
            AppointmentBORow newApp = FindMyAppointment((Guid)e.Appointment.Key[0]);
            AppointmentBORow editedApp = e.Appointment.Tag as AppointmentBORow;
            if (editedApp != null)
            {
                newApp.BOProperty1 = editedApp.BOProperty1;
                newApp.BOProperty2 = editedApp.BOProperty2;
            }
        }
        private void c1Scheduler1_UserEditingAppointment(object sender,
C1.Silverlight.Schedule.AppointmentActionEventArgs e)
        {
            AppointmentBORow myApp = FindMyAppointment((Guid)e.Appointment.Key[0]);
            e.Appointment.Tag = myApp;
        }
        AppointmentBORow FindMyAppointment(Guid id)
        {
            foreach (AppointmentBORow app in myAppointments.Appointments)
            {
                if (app.Id == id)
                    return app;
            }
            return null;
        }
    }
}
```

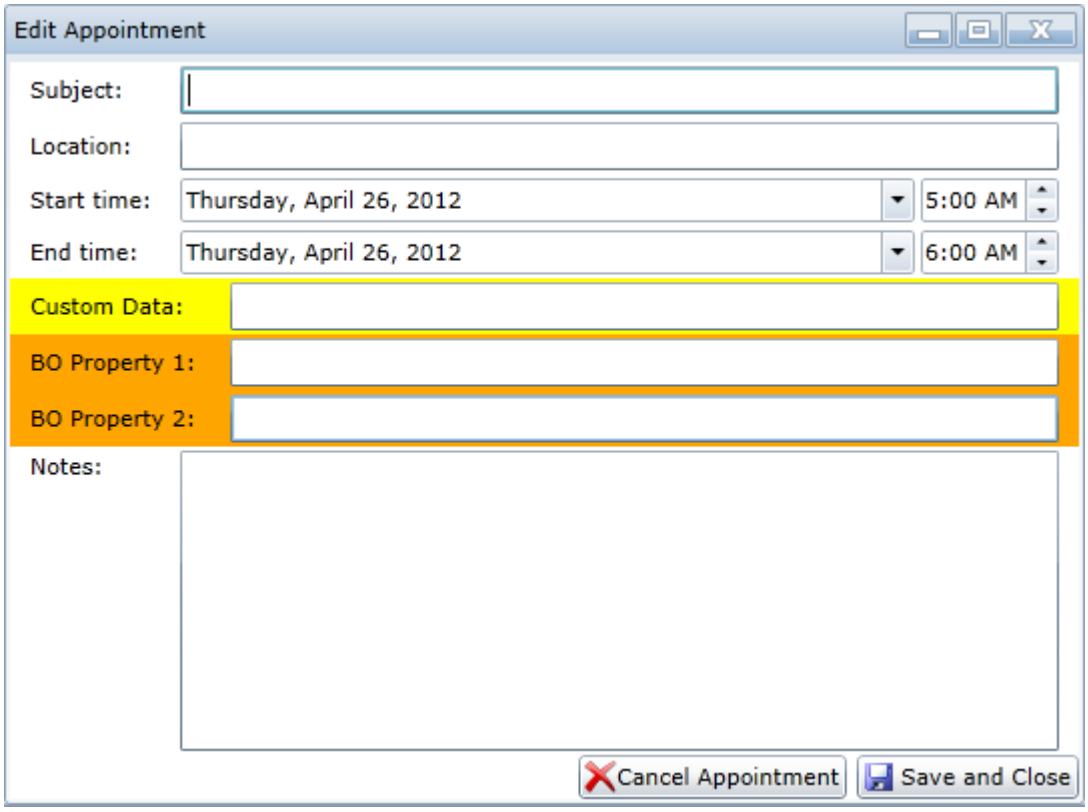
In this step you added the C1Scheduler control to your application and set all required properties for working with custom dialog and data.

Step 5 of 5: Completing the Application

In this step, you will complete and run the **Scheduler for Silverlight** application.

1. Right-click on the application name and select **Add | New Folder** from the list. Name the new folder **Images**.
2. Right-click on the **Images** folder and select **Add | Existing Item** from the list. Locate the images you would like to use for the **Save and Close Button** and the **Cancel Appointment Button** and add them to the folder.
3. Select the Image control on the **Save and Close Button**. In the Properties window, locate the **Source** property. Click the ellipsis button to open the **Choose Image** dialog and select the image you would like to use for the **Save and Close Button**. Repeat this step to set the image for the **Cancel Appointment Button**.
4. Press **F5** or start debugging to run the application. The application should initially appear as follows:

5. Double-click on an appointment time to open the **Edit Appointment** dialog box:



What You've Accomplished:

You created a C1Scheduler control and a custom **Edit Appointment** dialog box. You created data bindings for the **Edit Appointment** dialog box fields and added images to the dialog buttons.

Creating a Custom View

You can easily create a custom view for your C1Scheduler control. In this tutorial you will create a new Silverlight application, set Brush Styles and Data Templates in the MainPage.xaml file, and add code to control the **button_Click()** events.

Step 1 of 4: Creating the Application

In this step you will create the Silverlight application and add the appropriate references and namespaces. You will also set the Brush Styles in your MainPage.xaml file.

Follow these steps:

1. Create a new Silverlight application in Visual Studio and name it **CustomView**.
2. Add the following references to your application by right-clicking the References folder in the Solution Explorer and selecting **Add Reference** from the list. Browse to the folder where your references are located and select the following:
    - C1.Silverlight
    - C1.Silverlight.Data
    - C1.Silverlight.DateTimeEditors

- C1.Silverlight.Schedule

3. In your **MainPage.xaml** file, add the following to the namespace declarations:

   xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"

4. Add the *<UserControl.Resources>* *<UserControl.Resources/>* tags below the namespace declarations.
5. Click between the *<UserControl.Resources>* tags and insert the following XAML to provide the style and brush resources for the page:

XMAL

```
<!-- Border Brush -->
    <LinearGradientBrush x:Key="BorderBrush" EndPoint="0.5,1"
StartPoint="0.5,0">
        <GradientStop Color="#FFA3AEB9" Offset="0"/>
        <GradientStop Color="#FF8399A9" Offset="0.375"/>
        <GradientStop Color="#FF718597" Offset="0.375"/>
        <GradientStop Color="#FF617584" Offset="1"/>
    </LinearGradientBrush>

    <!-- C1Menu styles-->
    <ControlTemplate TargetType="c1:C1Separator"
x:Key="FirstLevelSeparatorTemplate">
        <Grid x:Name="Root" Background="#FF3C688D" Opacity="0.2"
Margin="2,5,5,8" Width="0.5"/>
    </ControlTemplate>
        <Style TargetType="c1:C1Separator" x:Key="Separator">
        <Setter Property="FirstLevelTemplate" Value="{StaticResource
FirstLevelSeparatorTemplate}"/>
        </Style>

    <!-- C1Scheduler brushes -->
    <!-- Current day header -->
    <LinearGradientBrush x:Key="C1Scheduler_TodayHeader_Brush" EndPoint="0.5,1"
StartPoint="0.5,0">
        <GradientStop Color="#FFF6CE71" Offset="0"/>
        <GradientStop Color="#FFF5C664" Offset="0.5"/>
        <GradientStop Color="#FFEF9F26" Offset="0.5"/>
        <GradientStop Color="#FFF4BE59" Offset="1"/>
    </LinearGradientBrush>
    <LinearGradientBrush x:Key="DayHeader_HoverBrush" EndPoint="0.5,1"
StartPoint="0.5,0">
        <GradientStop Color="#FFFFFCDE" Offset="0"/>
        <GradientStop Color="#FFFFEAAC" Offset="0.5"/>
        <GradientStop Color="#FFFFD767" Offset="0.5"/>
        <GradientStop Color="#FFFFE59B" Offset="1"/>
    </LinearGradientBrush>
    <LinearGradientBrush x:Key="TodayHeader_HoverBrush" EndPoint="0.5,1"
StartPoint="0.5,0">
        <GradientStop Color="#FFFFFCDE" Offset="0"/>
        <GradientStop Color="#FFFFEAAC" Offset="0.5"/>
        <GradientStop Color="#FFFFD767" Offset="0.5"/>
        <GradientStop Color="#FFFFE59B" Offset="1"/>
    </LinearGradientBrush>
```

Step 2 of 4: Creating the Data Templates and Adding C1Scheduler

In this step you will create the Data Templates for the different C1Scheduler views and the **C1Schedule** and **C1Calendar** controls that will make up your main view.

1. Add the following markup after the style and brush resources to create the C1Scheduler Converter and the Data Template for displaying a single day:

XAML

```
<!-- C1Scheduler converters -->
    <c1:DateTimeToStringConverter x:Key="DateTimeToStringConverter"/>
    <c1:BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>
    <c1:VisualIntervalToStringConverter x:Key="VisualIntervalToStringConverter"/>

    <!-- determines the style used for displaying single day (selects different styles for current day and other
days -->
    <c1:DayIntervalStyleSelector x:Key="DayStyleSelector">
        <c1:DayIntervalStyleSelector.Resources>
            <ResourceDictionary>

            <DataTemplate x:Key="DayHeaderTemplate">
                <Button HorizontalContentAlignment="Stretch" VerticalContentAlignment="Center"
                            c1:CommandExtensions.CommandParameter="{Binding Path=Scheduler.OneDayStyle}"
                        c1:CommandExtensions.Command="c1:C1Scheduler.ChangeStyleCommand"
                            c1:CommandExtensions.CommandTarget="{Binding Path=Scheduler}">
                    <Button.Style>
                        <Style TargetType="Button">
                            <Setter Property="Template">
                                <Setter.Value>
                                    <ControlTemplate TargetType="Button">
                                        <Grid>
                                            <VisualStateManager.VisualStateGroups>
                                                <VisualStateGroup x:Name="CommonStates">
                                                    <VisualState x:Name="Normal" />
                                                    <VisualState x:Name="MouseOver">
                                                        <Storyboard>
                                                            <ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
                                                                        Duration="00:00:00.0010000"
                                                                        Storyboard.TargetName="DayHeaderBorder"
                                                                        Storyboard.TargetProperty="Background">
                                                                <DiscreteObjectKeyFrame KeyTime="00:00:00"
Value="{StaticResource DayHeader_HoverBrush}"/>
                                                            </ObjectAnimationUsingKeyFrames>
                                                        </Storyboard>
                                                    </VisualState>
                                                </VisualStateGroup>
                                            </VisualStateManager.VisualStateGroups>
                                            <Border x:Name="DayHeaderBorder" BorderBrush="#FF5A8ECE" VerticalAlignment="Stretch"
BorderThickness="0,0,0,1" Margin="0">
                                                <Border.Background>
                                                    <LinearGradientBrush EndPoint="0.5,1" StartPoint="0.5,0">
                                                        <GradientStop Color="#FFE7EEF7" Offset="0"/>
                                                        <GradientStop Color="#FFD9E5F4" Offset="0.5"/>
                                                        <GradientStop Color="#FFC9D9EE" Offset="0.5"/>
                                                        <GradientStop Color="#FFE6EDF7" Offset="1"/>
                                                    </LinearGradientBrush>
                                                </Border.Background>
                                                <ContentPresenter Content="{TemplateBinding Content}" HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" />
                                            </Border>
                                        </Grid>
                                    </ControlTemplate>
                                </Setter.Value>
                            </Setter>
                        </Style>
                    </Button.Style>
                </c1:C1DockPanel>
                <TextBlock c1:C1DockPanel.Dock="Right" Margin="2" FontWeight="Semibold" Text="{Binding
Converter={StaticResource VisualIntervalToStringConverter},
```

```
                    ConverterParameter='ddd MMMM d, yyy'}"/>
            <!-- Add empty block to fill panel (it's needed for command) -->
                    <TextBlock />
                   </c1:C1DockPanel>
        </Button>
    </DataTemplate>
```

2. Insert the following XAML to create the Data Template for the Today Header:

XAML

```
<DataTemplate x:Key="TodayHeaderTemplate">
        <Button HorizontalContentAlignment="Stretch" VerticalContentAlignment="Center"
               c1:CommandExtensions.CommandParameter="{Binding Path=Scheduler.OneDayStyle}"
               c1:CommandExtensions.Command="c1:C1Scheduler.ChangeStyleCommand"
               c1:CommandExtensions.CommandTarget="{Binding Path=Scheduler}">
                <Button.Style>
                    <Style TargetType="Button">
                        <Setter Property="Template">
                            <Setter.Value>
                                <ControlTemplate TargetType="Button">
                                    <Grid>
                                        <VisualStateManager.VisualStateGroups>
                                            <VisualStateGroup x:Name="CommonStates">
                                                <VisualState x:Name="Normal" />
                                                    <VisualState x:Name="MouseOver">
                                                        <Storyboard>
                                                            <ObjectAnimationUsingKeyFrames BeginTime="00:00:00"
                                                                                           Duration="00:00:00.0010000"

Storyboard.TargetName="DayHeaderBorder"

Storyboard.TargetProperty="Background">
                                                                <DiscreteObjectKeyFrame KeyTime="00:00:00"
Value="{StaticResource TodayHeader_HoverBrush}"/>
                                                            </ObjectAnimationUsingKeyFrames>
                                                        </Storyboard>
                                                    </VisualState>
                                            </VisualStateGroup>
                                        </VisualStateManager.VisualStateGroups>
                                        <Border BorderThickness="0,0,0,1" x:Name="DayHeaderBorder"
                                                                          BorderBrush="Orange"
Background="{StaticResource C1Scheduler_TodayHeader_Brush}"
                                                VerticalAlignment="Stretch" Margin="0">
                                            <ContentPresenter Content="{TemplateBinding Content}"
                                                                       HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" />
                                                </Border>
                                            </Grid>
                                        </ControlTemplate>
                                    </Setter.Value>
                                </Setter>
                            </Style>
                        </Button.Style>
                        <c1:C1DockPanel>
                            <TextBlock c1:C1DockPanel.Dock="Right" Margin="2" FontWeight="Semibold"
                                   Text="{Binding Converter={StaticResource VisualIntervalToStringConverter},
                                   ConverterParameter='ddd MMMM d, yyy'}"/>
    <!-- Add empty block to fill panel (it's needed for command) -->
                            <TextBlock />
                        </c1:C1DockPanel>
                    </Button>
                </DataTemplate>
```

3. The XAML below will create the styles for the Day Header and the Today Header:

XAML

```
<!-- determines the style of the day header. It is used in all Office 2007 views -->
                    <Style x:Key="C1Scheduler_Day_Style" TargetType="c1:C1ListBoxItem">
                        <Setter Property="BorderThickness" Value="0,0,1,1" />
                        <Setter Property="HorizontalContentAlignment" Value="Stretch" />
                        <Setter Property="VerticalContentAlignment" Value="Stretch" />
                        <Setter Property="UseLayoutRounding" Value="true" />
```

```
                        <Setter Property="Template">
                            <Setter.Value>
                                <ControlTemplate TargetType="c1:C1ListBoxItem">
                                    <Border x:Name="DayBorder"
                                       BorderThickness="{TemplateBinding BorderThickness}"
                                       BorderBrush="{TemplateBinding BorderBrush}">
                                        <c1:C1DockPanel >
                                            <ContentPresenter Content="{TemplateBinding Content}"
                                                        c1:C1DockPanel.Dock="Top"
                                                        ContentTemplate="{StaticResource DayHeaderTemplate}" />
                                            <ContentPresenter Content="{TemplateBinding Content}"
                                                        ContentTemplate="{TemplateBinding ContentTemplate}"
                                                                  HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" />
                                        </c1:C1DockPanel>
                                    </Border>
                                </ControlTemplate>
                            </Setter.Value>
                        </Setter>
                    </Style>

    <!-- determines the style of the today header. It is used in all Office 2007 views -->
                    <Style x:Key="C1Scheduler_Today_Style" TargetType="c1:C1ListBoxItem">
                        <Setter Property="BorderThickness" Value="1,0,1,1" />
                        <Setter Property="HorizontalContentAlignment" Value="Stretch" />
                        <Setter Property="VerticalContentAlignment" Value="Stretch" />
                        <Setter Property="BorderBrush" Value="Orange" />
                        <Setter Property="UseLayoutRounding" Value="true" />
                        <Setter Property="Template">
                            <Setter.Value>
                                <ControlTemplate TargetType="c1:C1ListBoxItem">
                                    <Border x:Name="DayBorder"
                                       BorderThickness="{TemplateBinding BorderThickness}"
                                       BorderBrush="{TemplateBinding BorderBrush}" Margin="-1 0 0 1">
                                        <c1:C1DockPanel>
                                            <ContentPresenter Content="{TemplateBinding Content}"
                                                        c1:C1DockPanel.Dock="Top"
                                                        ContentTemplate="{StaticResource TodayHeaderTemplate}" />
                                            <ContentPresenter Content="{TemplateBinding Content}"
                                                      ContentTemplate="{TemplateBinding ContentTemplate}"
                                                              HorizontalAlignment="Stretch"
VerticalAlignment="Stretch" />
                                        </c1:C1DockPanel>
                                    </Border>
                                </ControlTemplate>
                            </Setter.Value>
                        </Setter>
                    </Style>
                </ResourceDictionary>
            </c1:DayIntervalStyleSelector.Resources>
        </c1:DayIntervalStyleSelector>
```

4. Insert the following XAML markup to set the styles for the Week Style, the Two Week Style, and the Work Week Style:

XAML

```
<!-- Custom Week Style -->
        <Style x:Key="CustomWeekStyle" TargetType="c1:C1Scheduler" >
            <Setter Property="c1:C1Scheduler.VisualTimeSpan" Value="7" />
            <Setter Property="c1:C1Scheduler.VisualIntervalScale" Value="1" />
            <Setter Property="FontSize" Value="11" />
            <Setter Property="FontFamily" Value="Segoe UI"/>
            <Setter Property="Background" Value="#FFBDD7F2"/>
            <Setter Property="BorderBrush" Value="Violet"/>
            <Setter Property="AppointmentForeground" Value="Navy"/>
            <Setter Property="SmallStartTimeChange" Value="7" />
            <Setter Property="LargeStartTimeChange" Value="7" />
            <Setter Property="c1:C1Scheduler.Template">
                <Setter.Value>
                    <ControlTemplate TargetType="c1:C1Scheduler">
                        <Border BorderBrush="{TemplateBinding BorderBrush}"
                                    BorderThickness="{TemplateBinding BorderThickness}">
    <!-- set DataContext to the first visible group item -->
```

```
                            <Grid Name="grid" DataContext="{Binding VisibleGroupItems[0], RelativeSource=
{RelativeSource TemplatedParent}}">
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition />
                                    <ColumnDefinition Width="Auto" />
                                </Grid.ColumnDefinitions>
    <!-- nested setters should present in the visual tree, in other case bindings might not work for them -->
                                <ItemsControl ItemsSource="{Binding RelativeSource={RelativeSource
TemplatedParent}, Path=NestedSetters}" IsHitTestVisible="False"/>
                                <c1:C1UniformGrid Grid.Column="0" Rows="4" Columns="2" Background="{TemplateBinding
Background}" >
                                    <Border Background="#FFE7EFFF" BorderBrush="#FF6392CE" BorderThickness="0 0 1
1">
                                        <c1:C1WrapPanel Margin="2,10,2,2"
                                                        VerticalAlignment="Center" HorizontalAlignment="Center">
                                            <TextBlock FontSize="12"
                                                Text="{Binding VisualIntervals[0],
                                                Converter={StaticResource VisualIntervalToStringConverter},
                                                ConverterParameter=D}" Margin="2"/>
                                            <TextBlock FontSize="12" Text=" - " Margin="2"/>
                                            <TextBlock FontSize="12"
                                                Text="{Binding Path=VisualIntervals[6],
                                                Converter={StaticResource VisualIntervalToStringConverter},
                                                ConverterParameter=D}" Margin="2"/>
                                        </c1:C1WrapPanel>
                                    </Border>
                                </c1:C1UniformGrid>
                                <c1:C1SchedulerPresenter Grid.Column="0"
                                     ItemContainerStyleSelector="{StaticResource DayStyleSelector}"/>
                                <c1:AppointmentsCoverPane Grid.Column="0"
                                                          IsDragDropDisabled="{Binding RelativeSource={RelativeSource
TemplatedParent},
                                                          Path=IsDragDropDisabled}" CoverElementsMargin="10" />
                                <ScrollBar x:Name="PART_C1SchedulerScrollBar" Grid.Column="1"
Orientation="Vertical"/>
                            </Grid>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="c1:C1Scheduler.VisualIntervalPanel">
                <Setter.Value>
                    <DataTemplate>
                        <c1:C1UniformGrid Rows="4" Columns="2" FirstColumn="1" />
                    </DataTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="c1:C1Scheduler.VisualIntervalTemplate">
                <Setter.Value>
                    <DataTemplate>
                        <Grid>
                            <Border Name="DayContentBorder" UseLayoutRounding="True"
                                    c1:AppointmentsCoverPane.KeepTimesAtDrop="True"
                                    c1:CoverElementsPane.Orientation="Horizontal"
                                Background="White">
                                <Border Name="highlightStatusBorder" Background="{Binding Path=StatusBrush}"
Opacity="0.1"/>
                            </Border>
                            <Border Background="#4090b3de"
                                Visibility="{Binding IsSelected, Converter={StaticResource
BooleanToVisibilityConverter}}">
                                </Border>
                        </Grid>
                    </DataTemplate>
                </Setter.Value>
            </Setter>
        </Style>

    <!-- Two Week Style -->
        <Style x:Key="TwoWeekStyle" TargetType="c1:C1Scheduler" >
            <Setter Property="c1:C1Scheduler.VisualTimeSpan" Value="14" />
            <Setter Property="c1:C1Scheduler.VisualIntervalScale" Value="1" />
```

```
            <Setter Property="FontSize" Value="11" />
            <Setter Property="FontFamily" Value="Segoe UI"/>
            <Setter Property="AppointmentForeground" Value="Navy"/>
            <Setter Property="Background" Value="#FFBDD7F2"/>
            <Setter Property="BorderBrush" Value="Transparent"/>
            <Setter Property="SmallStartTimeChange" Value="7" />
            <Setter Property="LargeStartTimeChange" Value="14" />
            <Setter Property="c1:C1Scheduler.Template">
                <Setter.Value>
                    <ControlTemplate TargetType="c1:C1Scheduler">
                        <Border BorderBrush="{TemplateBinding BorderBrush}" Background="{TemplateBinding
Background}"
                                            BorderThickness="{TemplateBinding BorderThickness}">
    <!-- set DataContext to the first visible group item -->
                            <Grid Name="grid" DataContext="{Binding VisibleGroupItems[0], RelativeSource=
{RelativeSource TemplatedParent}}">
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition />
                                    <ColumnDefinition Width="Auto" />
                                </Grid.ColumnDefinitions>
                                <!-- nested setters should present in the visual tree, in other case bindings might
not work for them -->
                                <ItemsControl ItemsSource="{Binding RelativeSource={RelativeSource
TemplatedParent}, Path=NestedSetters}" IsHitTestVisible="False"/>
                                <c1:C1SchedulerPresenter Grid.Column="0"
                                     ItemContainerStyleSelector="{StaticResource DayStyleSelector}"/>
                                <c1:AppointmentsCoverPane Grid.Column="0"
                                                IsDragDropDisabled="{Binding RelativeSource={RelativeSource
TemplatedParent},
                                                Path=IsDragDropDisabled}" CoverElementsMargin="10" />
                                <ScrollBar x:Name="PART_C1SchedulerScrollBar" Grid.Column="1"
Orientation="Vertical"/>
                            </Grid>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="c1:C1Scheduler.VisualIntervalPanel">
                <Setter.Value>
                    <DataTemplate>
                        <c1:C1UniformGrid Rows="2" Columns="7" Background="{Binding Scheduler.Background}" />
                    </DataTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="c1:C1Scheduler.VisualIntervalTemplate">
                <Setter.Value>
                    <DataTemplate>
                        <Grid>
                            <Border Name="DayContentBorder" UseLayoutRounding="True"
                                            c1:AppointmentsCoverPane.KeepTimesAtDrop="True"
                                            c1:CoverElementsPane.Orientation="Horizontal"
                                    Background="White">
                                <Border Name="highlightStatusBorder" Background="{Binding Path=StatusBrush}"
Opacity="0.1"/>
                            </Border>
                            <Border Background="#4090b3de"
                                Visibility="{Binding IsSelected, Converter={StaticResource
BooleanToVisibilityConverter}}">
                            </Border>
                        </Grid>
                    </DataTemplate>
                </Setter.Value>
            </Setter>
        </Style>

    <!-- Custom Working Week Style -->
        <Style x:Key="CustomWorkingWeekStyle" TargetType="c1:C1Scheduler" >
            <Setter Property="c1:C1Scheduler.VisualTimeSpan" Value="7" />
            <Setter Property="c1:C1Scheduler.VisualIntervalScale" Value="1" />
            <Setter Property="Background" Value="#FFBDD7F2"/>
            <Setter Property="BorderBrush" Value="Transparent"/>
            <Setter Property="FontSize" Value="11" />
```

```
            <Setter Property="AppointmentForeground" Value="Navy"/>
            <Setter Property="FontFamily" Value="Segoe UI"/>
            <Setter Property="SmallStartTimeChange" Value="7" />
            <Setter Property="LargeStartTimeChange" Value="7" />
            <Setter Property="c1:C1Scheduler.Template">
                <Setter.Value>
                    <ControlTemplate TargetType="c1:C1Scheduler">
                        <Border BorderBrush="{StaticResource BorderBrush}" Background="#FF91BAEE"
                                        BorderThickness="{TemplateBinding BorderThickness}">
    <!-- set DataContext to the first visible group item -->
                            <Grid Name="grid" DataContext="{Binding VisibleGroupItems[0], RelativeSource=
{RelativeSource TemplatedParent}}">
                                <Grid.ColumnDefinitions>
                                    <ColumnDefinition />
                                    <ColumnDefinition Width="Auto" />
                                </Grid.ColumnDefinitions>
                                <!-- nested setters should present in the visual tree, in other case bindings might
not work for them -->
                                <ItemsControl ItemsSource="{Binding RelativeSource={RelativeSource
TemplatedParent}, Path=NestedSetters}" IsHitTestVisible="False"/>
                                <c1:C1SchedulerPresenter Grid.Column="0"
                                     ItemContainerStyleSelector="{StaticResource DayStyleSelector}"/>
                                <c1:AppointmentsCoverPane Grid.Column="0"
                                                IsDragDropDisabled="{Binding RelativeSource={RelativeSource
TemplatedParent},
                                                Path=IsDragDropDisabled}" CoverElementsMargin="10" />
                                <ScrollBar x:Name="PART_C1SchedulerScrollBar" Grid.Column="1"
Orientation="Vertical"/>
                            </Grid>
                        </Border>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="c1:C1Scheduler.VisualIntervalPanel">
                <Setter.Value>
                    <DataTemplate>
                        <c1:C1UniformGrid Rows="1"/>
                    </DataTemplate>
                </Setter.Value>
            </Setter>
            <Setter Property="c1:C1Scheduler.VisualIntervalTemplate">
                <Setter.Value>
                    <DataTemplate>
                        <Grid>
                            <Border Name="DayContentBorder" UseLayoutRounding="True"
                                            c1:AppointmentsCoverPane.KeepTimesAtDrop="True"
                                            c1:CoverElementsPane.Orientation="Horizontal"
BorderBrush="#FFBDD7F2"  BorderThickness="0,0,0,10"
                                        Background="White">
                                <Border Name="highlightStatusBorder" Background="{Binding Path=StatusBrush}"
Opacity="0.1"/>
                            </Border>
                            <Border Background="#4090b3de"
                                Visibility="{Binding IsSelected, Converter={StaticResource
BooleanToVisibilityConverter}}">
                            </Border>
                        </Grid>
                    </DataTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </UserControl.Resources>
```

5. Add the following XAML below the ending </UserControl.Resources> tag to create the C1Scheduler control and the C1Calendar controls:

XAML

```
<Grid x:Name="LayoutRoot">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
        </Grid.RowDefinitions>
        <c1:C1Menu Grid.Row="0" FontFamily="Segoe UI" FontSize="12" BorderThickness="1">
```

```
            <c1:C1MenuItem x:Name="btnToday" Header="Today"
                           ToolTipService.ToolTip="Navigate to the current date."
                           Click="btnToday_Click" />
            <c1:C1Separator Style="{StaticResource Separator}"/>
            <c1:C1MenuItem Header="View"
                           ToolTipService.ToolTip="Change current view.">
                <c1:C1MenuItem x:Name="btnDay" Header="Day"
                               Click="btnDay_Click"/>
                <c1:C1MenuItem x:Name="btnWorkWeek" Header="Working Week"
                               Click="btnWorkWeek_Click"/>
                <c1:C1MenuItem x:Name="btnWeek" Header="Week"
                               Click="btnWeek_Click"/>
                <c1:C1MenuItem x:Name="btn2Week" Header="Two Weeks"
                               Click="btn2Week_Click">
                </c1:C1MenuItem>
                <c1:C1MenuItem x:Name="btnMonth" Header="Month"
                               Click="btnMonth_Click"/>
                <c1:C1MenuItem x:Name="btnTimeLine" Header="Time Line"
                               Click="btnTimeLine_Click"/>
            </c1:C1MenuItem>
            <c1:C1Separator Style="{StaticResource Separator}" />
            <c1:C1MenuItem x:Name="btnImport" Header="Import..."
                           ToolTipService.ToolTip="Import data from the local file."
                           Click="btnImport_Click" />
            <c1:C1MenuItem x:Name="btnExport" Header="Export..."
                           ToolTipService.ToolTip="Export data to the local file."
                           Click="btnExport_Click" />
        </c1:C1Menu>
        <Grid Grid.Row="1">
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="192"/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <c1:C1Calendar x:Name="cal1" Grid.Column="0"
                           CalendarHelper="{Binding CalendarHelper, ElementName=sched1, Mode=OneWay}"
                                SelectedDates="{Binding VisibleDates, ElementName=sched1, Mode=OneWay}"
                                      BoldedDates="{Binding BoldedDates, ElementName=sched1, Mode=OneWay}"
                           MaxSelectionCount="42" GenerateAdjacentMonthDays="True" />
            <c1:C1Scheduler x:Name="sched1"
                           BeforeViewChange="sched1_BeforeViewChange"
                           BorderThickness="1" Grid.Column="1"
                           BorderBrush="{StaticResource BorderBrush}"
                           WorkingWeekStyle="{StaticResource CustomWorkingWeekStyle}"
                           WeekStyle="{StaticResource CustomWeekStyle}"/>
        </Grid>
    </Grid>
</UserControl>
```

Step 3 of 4: Adding the Code to Control the Application

In this step you will add the C# or Visual Basic code that will control the application.

1. Right-click the application and choose **View Code** from the list. Add the following namespace to the top of the page:

Visual Basic

```
Imports C1.Silverlight.Schedule
```

C#

```
using C1.Silverlight.Schedule;
```

2. Add the following **IDisposable** Implementation to the Public Partial Class:

Visual Basic

```
Partial Public Class MainPage
   Inherits UserControl
         Implements IDisposable
```

C#

```
public partial class MainPage : UserControl, IDisposable
```

3. Insert the following code directly under the **InitializeComponent()** method:

Visual Basic

```
sched1.ChangeStyle(sched1.WeekStyle)
    End Sub
    Private Sub btnToday_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        sched1.VisibleDates.BeginUpdate()
        sched1.VisibleDates.Clear()
        sched1.VisibleDates.Add(DateTime.Today)
        sched1.VisibleDates.EndUpdate()
    End Sub
    ' Change view from Month View mode to 2 Week mode if selection
    ' contains 2 full weeks.
    Private Sub sched1_BeforeViewChange(sender As Object, e As
BeforeViewChangeEventArgs)
        If e.Dates.Length = 14 AndAlso e.Style Is sched1.MonthStyle Then
            Dim s As Style = DirectCast(Me.Resources("TwoWeekStyle"), Style)
            If s IsNot Nothing Then
                e.Style = s
            End If
        End If
    End Sub
    Private Sub btnDay_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        sched1.ChangeStyle(sched1.OneDayStyle)
    End Sub
    Private Sub btnWorkWeek_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        sched1.ChangeStyle(sched1.WorkingWeekStyle)
    End Sub
    Private Sub btnWeek_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        sched1.ChangeStyle(sched1.WeekStyle)
    End Sub
    Private Sub btn2Week_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        sched1.ChangeStyle(Resources("TwoWeekStyle"))
    End Sub
    Private Sub btnMonth_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        sched1.ChangeStyle(sched1.MonthStyle)
    End Sub
    Private Sub btnTimeLine_Click(sender As Object, e As
```

```
C1.Silverlight.SourcedEventArgs)
        sched1.ChangeStyle(sched1.TimeLineStyle)
    End Sub
    Private Sub btnImport_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        C1Scheduler.ImportCommand.Execute(Nothing, sched1)
    End Sub
    Private Sub btnExport_Click(sender As Object, e As
C1.Silverlight.SourcedEventArgs)
        C1Scheduler.ExportCommand.Execute(Nothing, sched1)
    End Sub
#Region "IDisposable Support"
    Private disposedValue As Boolean ' To detect redundant calls
    ' IDisposable
    Protected Overridable Sub Dispose(disposing As Boolean)
        If Not Me.disposedValue Then
            If disposing Then
                ' TODO: dispose managed state (managed objects).
            End If
            ' TODO: free unmanaged resources (unmanaged objects) and override
Finalize() below.
            ' TODO: set large fields to null.
        End If
        Me.disposedValue = True
    End Sub
```

C#

```
sched1.ChangeStyle(sched1.WeekStyle);
        }
        private void btnToday_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            sched1.VisibleDates.BeginUpdate();
            sched1.VisibleDates.Clear();
            sched1.VisibleDates.Add(DateTime.Today);
            sched1.VisibleDates.EndUpdate();
        }
        // Change view from Month View mode to 2 Week mode if selection
        // contains 2 full weeks.
        void sched1_BeforeViewChange(object sender, BeforeViewChangeEventArgs e)
        {
            if (e.Dates.Length == 14 && e.Style == sched1.MonthStyle)
            {
                Style s = (Style)this.Resources["TwoWeekStyle"];
                if (s != null)
                {
                    e.Style = s;
                }
            }
        }
        private void btnDay_Click(object sender, C1.Silverlight.SourcedEventArgs
```

```
e)
        {
            sched1.ChangeStyle(sched1.OneDayStyle);
        }

        private void btnWorkWeek_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            sched1.ChangeStyle(sched1.WorkingWeekStyle);
        }
        private void btnWeek_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            sched1.ChangeStyle(sched1.WeekStyle);
        }
        private void btn2Week_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            sched1.ChangeStyle(Resources["TwoWeekStyle"]);
        }
        private void btnMonth_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            sched1.ChangeStyle(sched1.MonthStyle);
        }
        private void btnTimeLine_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            sched1.ChangeStyle(sched1.TimeLineStyle);
        }
        private void btnImport_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            C1Scheduler.ImportCommand.Execute(null, sched1);
        }
        private void btnExport_Click(object sender,
C1.Silverlight.SourcedEventArgs e)
        {
            C1Scheduler.ExportCommand.Execute(null, sched1);
        }
```

4. Add the code below to handle the **IDisposable** implementation. The Visual Basic code should be added automatically by Visual Studio:

Visual Basic

```
#Region "IDisposable Support"
    Private disposedValue As Boolean ' To detect redundant calls
    ' IDisposable
    Protected Overridable Sub Dispose(disposing As Boolean)
        If Not Me.disposedValue Then
            If disposing Then
```

```
                sched1.Dispose()
            End If
            ' TODO: free unmanaged resources (unmanaged objects) and override
Finalize() below.
            ' TODO: set large fields to null.
        End If
        Me.disposedValue = True
    End Sub
    ' TODO: override Finalize() only if Dispose(ByVal disposing As Boolean)
above has code to free unmanaged resources.
    'Protected Overrides Sub Finalize()
    '    ' Do not change this code.  Put cleanup code in Dispose(ByVal disposing
As Boolean) above.
    '    Dispose(False)
    '    MyBase.Finalize()
    'End Sub
    ' This code added by Visual Basic to correctly implement the disposable
pattern.
    Public Sub Dispose() Implements IDisposable.Dispose
        ' Do not change this code.  Put cleanup code in Dispose(ByVal disposing
As Boolean) above.
        Dispose(True)
        GC.SuppressFinalize(Me)
    End Sub
```

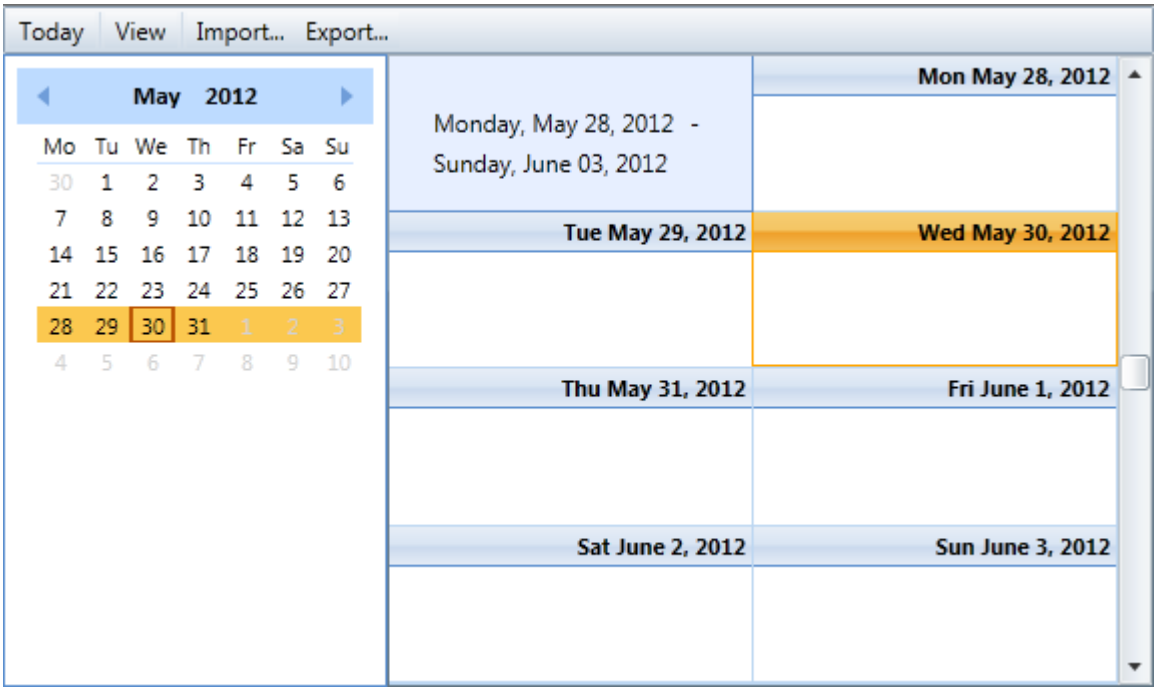
C#

```
public void Dispose()
        {
            sched1.Dispose();
        }
```

Step 4 of 4: Running the Application

This step completes the tutorial.

Press **F5** to run your application. The application should resemble the following image:



What You've Accomplished

You created a new Silverlight Application and a Custom Scheduler View using XAML markup and code.

Creating a Multi-User Schedule

In this tutorial, you will create an application that will allow you to display multiple schedules. You will use XAML markup and code to create the interface. You will also add a Data Service and two code files as resources for your application.

Step 1 of 4: Creating the Application

In this step, you will create a new Silverlight application, add the appropriate references and namespaces, and add two code files and a Data Service to the application.

Follow these steps:

1. Create a new Silverlight application in Visual Studio and name it **MultiUser**.
2. Add the following references to your application by right-clicking the References folder in the Solution Explorer and selecting **Add Reference** from the list. Browse to the folder where your references are located and select the following assemblies:
    - C1.Silverlight
    - C1.Silverlight.Data
    - C1.Silverlight.DateTimeEditors
    - C1.Silverlight.Schedule
3. Right-click the **MultiUser.Web** project and select **Add | Existing Item** from the list. Add the **SmartData** file that applies to your application (either the **SmartData.cs** or the **SmartData.vb** file) and the **PlatformUriTranslator** file that applies to your application (either the **PlatformUriTranslator.cs** or the **PlatformUriTranslator.vb** file).

4. Right-click **MultiUser.Web** again and select **Add | New Item** from the list. Add a basic Web Service and name it **DataService.asmx**.
5. Add the following code to the **DataService.asmx** file below the summary description for the web service:

Visual Basic

```
Public Class DataService
    Inherits System.Web.Services.WebService

    <WebMethod()> _
    Public Function GetData(tables As String) As Byte()
        ' Create DataSet with connection string
        Dim ds = GetDataSet()

        ' Load data into DataSet
        ds.Fill(tables.Split(","c))

        ' Persist to stream
        Dim ms = New System.IO.MemoryStream()
        ds.WriteXml(ms, XmlWriteMode.WriteSchema)

        ' Return stream data
        Return ms.ToArray()
    End Function

    <WebMethod()> _
    Public Function UpdateData(dtAdded As Byte(), dtModified As Byte(),
dtDeleted As Byte()) As String
        Try
            UpdateData(dtAdded, DataRowState.Added)
            UpdateData(dtModified, DataRowState.Modified)
            UpdateData(dtDeleted, DataRowState.Deleted)
            Return Nothing
        Catch x As Exception
            Return x.Message
        End Try
    End Function

    Private Sub UpdateData(data As Byte(), state As DataRowState)
        ' No changes, no work
        If data Is Nothing Then
            Return
        End If

        ' Load data into dataset
        Dim ds = GetDataSet()
        Dim ms = New MemoryStream(data)
        ds.ReadXml(ms)
        ds.AcceptChanges()

        ' Update row states with changes
        For Each dt As DataTable In ds.Tables
```

```
            For Each dr As DataRow In dt.Rows
                Select Case state
                    Case DataRowState.Added
                        dr.SetAdded()
                        Exit Select
                    Case DataRowState.Modified
                        dr.SetModified()
                        Exit Select
                    Case DataRowState.Deleted
                        dr.Delete()
                        Exit Select
                End Select
            Next
        Next

        ' Update the database
        ds.Update()
    End Sub

    Private Function GetDataSet() As SmartDataSet
        ' Get physical location of the mdb file
        Dim mdb As String =
Path.Combine(Context.Request.PhysicalApplicationPath, "App_Data\Nwind.mdb")

        ' Check that the file exists
        If Not File.Exists(mdb) Then
            Dim msg As String = String.Format("Cannot find database file {0}.",
mdb)
            Throw New FileNotFoundException(msg)
        End If

        ' Make sure file is not read-only (source control often does this...)
        Dim att As FileAttributes = File.GetAttributes(mdb)
        If (att And FileAttributes.[ReadOnly]) <> 0 Then
            att = att And Not FileAttributes.[ReadOnly]
            File.SetAttributes(mdb, att)
        End If

        ' Create and initialize the SmartDataSet
        Dim dataSet = New SmartDataSet()
        dataSet.ConnectionString = "provider=microsoft.jet.oledb.4.0;data
source=" & mdb

        ' sample connection string for SQL Server Express (replace Initial
Catalog value by the valid catalog name)
        ' dataSet.ConnectionString = "Provider=SQLOLEDB.1;Integrated
Security=SSPI;Persist Security Info=False;Initial Catalog=test;Data
Source=.\\SQLEXPRESS";
        Return dataSet
    End Function
End Class
```

C#

```
public class DataService : System.Web.Services.WebService
    {
        [WebMethod]

        public byte[] GetData(string tables)
        {
            // Create DataSet with connection string
            var ds = GetDataSet();

            // Load data into DataSet
            ds.Fill(tables.Split(','));

            // Persist to stream
            var ms = new System.IO.MemoryStream();
            ds.WriteXml(ms, XmlWriteMode.WriteSchema);

            // Return stream data
            return ms.ToArray();
        }

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

        void UpdateData(byte[] data, DataRowState state)
        {
            // No changes, no work
            if (data == null)
            {
                return;
            }

            // Load data into dataset
            var ds = GetDataSet();
            var ms = new MemoryStream(data);
            ds.ReadXml(ms);
            ds.AcceptChanges();
```

```
            // Update row states with changes
            foreach (DataTable dt in ds.Tables)
            {
                foreach (DataRow dr in dt.Rows)
                {
                    switch (state)
                    {
                        case DataRowState.Added:
                            dr.SetAdded();
                            break;
                        case DataRowState.Modified:
                            dr.SetModified();
                            break;
                        case DataRowState.Deleted:
                            dr.Delete();
                            break;
                    }
                }
            }

            // Update the database
            ds.Update();
        }

        SmartDataSet GetDataSet()
        {
            // Get physical location of the mdb file
            string mdb = Path.Combine(
              Context.Request.PhysicalApplicationPath, @"App_Data\Nwind.mdb");

            // Check that the file exists
            if (!File.Exists(mdb))
            {
                string msg = string.Format("Cannot find database file {0}.",
mdb);
                throw new FileNotFoundException(msg);
            }

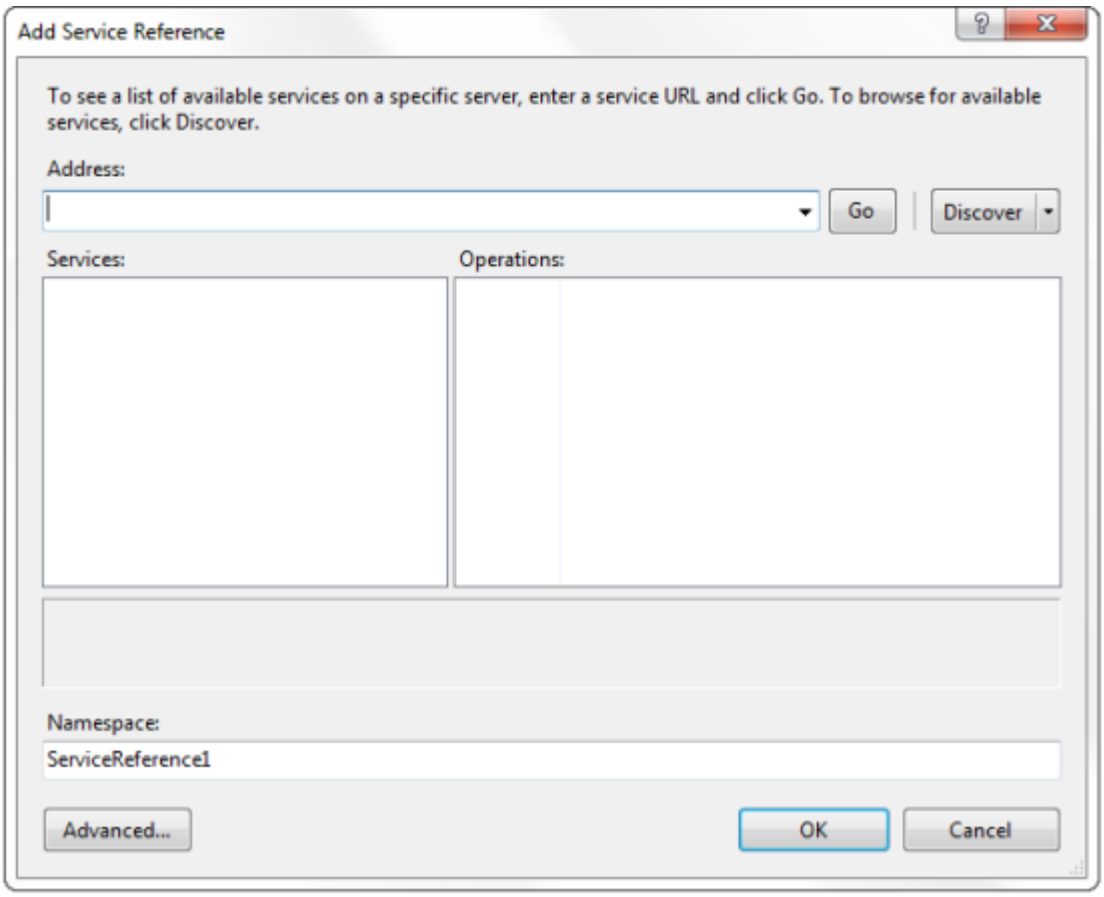
            // Make sure file is not read-only (source control often does
this...)
            FileAttributes att = File.GetAttributes(mdb);
            if ((att & FileAttributes.ReadOnly) != 0)
            {
                att &= ~FileAttributes.ReadOnly;
                File.SetAttributes(mdb, att);
            }

            // Create and initialize the SmartDataSet
            var dataSet = new SmartDataSet();
            dataSet.ConnectionString =
```

```
                "provider=microsoft.jet.oledb.4.0;data source=" + mdb;

            // sample connection string for SQL Server Express (replace Initial
Catalog value by the valid catalog name)
            // dataSet.ConnectionString = "Provider=SQLOLEDB.1;Integrated
Security=SSPI;Persist Security Info=False;Initial Catalog=test;Data
Source=.\\SQLEXPRESS";
            return dataSet;
        }
    }
}
```

6. Right-click **MultiUser.web** one more time and select **Add | Folder** from the list. Name the new folder **App_Data**.
7. Right-click the **App_Data** folder and select **Add | Existing Item** from the list. Locate and select the **Nwind.mdb** database.
8. Rebuild your entire application and then right-click your main **MultiUser** application. Select **Add Service Reference** from the list. The **Add Service Reference** dialog box will appear:



9. Click the **Discover** button to find the available service reference and then follow these steps:
    - Select the available service reference in the **Services** listbox.
    - Rename the service reference **DataService**.
    - Click **OK** to add the service reference to your application.

In this step you created a new Silverlight application, added the appropriate assembly references, and added a service reference and two code files. In the next step you will add XAML to your **MultiUser** application.

Step 2 of 4: Creating the Resources and the C1Scheduler Control

In this step you will create your application resources and data bindings. You will also add and customize a C1Scheduler control.

Follow these steps:

1. Add the following namespaces to your main **MultiUser** application:

XAML

```
xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
xmlns:c1sched="clr-
namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule"
Or, for Silverlight 5:
xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight.5"
xmlns:c1sched="clr-
namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule.5"
The entire namespace declaration should appear as follows:
<UserControl x:Class="MultipleUserTest.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:c1="clr-namespace:C1.Silverlight;assembly=C1.Silverlight"
xmlns:c1sched="clr-
namespace:C1.Silverlight.Schedule;assembly=C1.Silverlight.Schedule"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
mc:Ignorable="d"
d:DesignHeight="300" d:DesignWidth="400">
```

2. Add *<UserControl.Resources>* *</UserControl.Resources>* tags below the namespace declarations.
3. Click between the tags and insert the following Resource Dictionary:

XAML

```
<ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary
Source="/C1.Silverlight.Schedule;component/themes/generic.xaml"/>
            </ResourceDictionary.MergedDictionaries>
```

4. Add the following XAML after the Resource Dictionary to create the Control Template:

XAML

```
<ControlTemplate x:Key="looklessButton" TargetType="Button">
                <Border>
                    <ContentPresenter Margin="4,0" VerticalAlignment="Center"/>
                </Border>
            </ControlTemplate>
```

5. Create two custom header templates with the following XAML markup. This will create the Custom Group Header Template and the Custom Time Line Group Header Template:

XAML

```
<!-- custom GroupHeaderTemplate -->
            <DataTemplate x:Key="myCustomGroupHeaderTemplate">
                <Grid>
                    <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition/>
                        <ColumnDefinition Width="Auto"/>
                        <ColumnDefinition Width="Auto"/>
                    </Grid.ColumnDefinitions>
                    <Grid.RowDefinitions>
                        <RowDefinition />
                        <RowDefinition Height="25" />
                    </Grid.RowDefinitions>
                    <c1:C1BrushBuilder x:Name="Background" Style="{StaticResource
C1Scheduler_ControlArea_BrushStyle}" Input="{Binding Background}"/>
                    <c1:C1BrushBuilder x:Name="BorderBrush" Input="{Binding
Background}" Style="{StaticResource
C1Scheduler_MonthHeaderForeground_BrushStyle}"/>
                    <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}"/>
                    <!-- navigate to the first group -->
                    <Button Template="{StaticResource looklessButton}"
Content="|&lt;&lt;" Grid.Column="0" Grid.RowSpan="2" VerticalAlignment="Center"
                                                FontSize="12"

c1:CommandExtensions.Command="c1sched:C1Scheduler.NavigateToPreviousGroupCommand"

c1:CommandExtensions.CommandParameter="Home"

c1:CommandExtensions.CommandTarget="{Binding Scheduler}"
                                                Visibility="{Binding
ShowPreviousButton, Converter={StaticResource BooleanToVisibilityConverter}}"/>
                    <!-- navigate to the previous group -->
                    <Button Template="{StaticResource looklessButton}"
Content="&lt;" Grid.Column="1" Grid.RowSpan="2" VerticalAlignment="Center"
                                                FontSize="12"

c1:CommandExtensions.Command="c1sched:C1Scheduler.NavigateToPreviousGroupCommand"

c1:CommandExtensions.CommandTarget="{Binding Scheduler}"
                                                Visibility="{Binding
```

```
ShowPreviousButton, Converter={StaticResource BooleanToVisibilityConverter}}"/>
                    <!-- navigate to the next group -->
                    <Button Template="{StaticResource looklessButton}"
Content="&gt;" Grid.Column="3" Grid.RowSpan="2" VerticalAlignment="Center"
                                                    FontSize="12"

c1:CommandExtensions.Command="c1sched:C1Scheduler.NavigateToNextGroupCommand"

c1:CommandExtensions.CommandTarget="{Binding Scheduler}"
                                                    Visibility="{Binding
ShowNextButton, Converter={StaticResource BooleanToVisibilityConverter}}"/>
                    <!-- navigate to the last group -->
                    <Button Template="{StaticResource looklessButton}"
Content="&gt;&gt;|" Grid.Column="4" Grid.RowSpan="2" VerticalAlignment="Center"
                                                    FontSize="12"

c1:CommandExtensions.Command="c1sched:C1Scheduler.NavigateToNextGroupCommand"

c1:CommandExtensions.CommandParameter="End"

c1:CommandExtensions.CommandTarget="{Binding Scheduler}"
                                                    Visibility="{Binding
ShowNextButton, Converter={StaticResource BooleanToVisibilityConverter}}"/>
                    <TextBlock Foreground="{Binding Path=Scheduler.Foreground}"
Margin="10,3" Grid.Column="2"
                                                Visibility="{Binding IsSelected,
Converter={StaticResource BooleanToVisibilityConverter},
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={StaticResource
BooleanToVisibilityConverter}}"
                                                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"/>
                </Grid>
            </DataTemplate>

            <!-- use different group header for TimeLine style -->
            <DataTemplate x:Key="myCustomTimeLineGroupHeaderTemplate">
                <Grid IsHitTestVisible="False">
                    <c1:C1BrushBuilder x:Name="Background" Style="{StaticResource
C1Scheduler_ControlArea_BrushStyle}" Input="{Binding Background}"/>
                    <c1:C1BrushBuilder x:Name="BorderBrush"
Style="{StaticResource C1Scheduler_MonthHeaderForeground_BrushStyle}"
```

```
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>
```

6. Add the following XAML markup to complete the styles and databindings used in your application:

XAML

```
<!-- Border Brush -->
            <LinearGradientBrush x:Key="BorderBrush" EndPoint="0.5,1"
StartPoint="0.5,0">
                <GradientStop Color="#FFA3AEB9" Offset="0"/>
                <GradientStop Color="#FF8399A9" Offset="0.375"/>
                <GradientStop Color="#FF718597" Offset="0.375"/>
                <GradientStop Color="#FF617584" Offset="1"/>
            </LinearGradientBrush>

            <!-- Button styles -->
            <SolidColorBrush x:Key="commandBaseBorderBrush"  Color="#FF6593CF"/>
            <LinearGradientBrush x:Key="buttBackBrush" StartPoint="0,0"
EndPoint="0,1" >
                <GradientStop Color="#FFE3EFFF" Offset="0"/>
                <GradientStop Color="#FFC4DDFF" Offset="0.367"/>
                <GradientStop Color="#FFADD1FF" Offset="0.377"/>
                <GradientStop Color="#FFC0DBFF" Offset="1"/>
            </LinearGradientBrush>
            <Style x:Key="buttonStyle"  TargetType="Button">
                <Setter Property="Width" Value="90"/>
                <Setter Property="Background" Value="{StaticResource
buttBackBrush}" />
                <Setter Property="Foreground" Value="#FF000000"/>
                <Setter Property="HorizontalContentAlignment" Value="Center"/>
                <Setter Property="VerticalContentAlignment" Value="Center"/>
```

```
                <Setter Property="FontSize" Value="14"/>
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate TargetType="Button">
                            <Grid>
                                <VisualStateManager.VisualStateGroups>
                                    <VisualStateGroup x:Name="CommonStates">
                                        <VisualState x:Name="Normal"/>
                                        <VisualState x:Name="MouseOver">
                                            <Storyboard>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFFCDE"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFEAAC"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFD767"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#FFFFE59B"/>
                                            </Storyboard>
                                        </VisualState>
                                        <VisualState x:Name="Pressed">
                                            <Storyboard>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFFCDE"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFEAAC"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFD767"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="Orange"/>
                                            </Storyboard>
                                        </VisualState>
                                    </VisualStateGroup>
                                </VisualStateManager.VisualStateGroups>
                                <Border BorderThickness="0,0,1,0">
                                    <Border.Background>
                                        <LinearGradientBrush
x:Name="BackgroundGradient" StartPoint="0,0" EndPoint="0,1" >
                                            <GradientStop Color="#FFE3EFFF"
Offset="0"/>
                                            <GradientStop Color="#FFC4DDFF"
Offset="0.367"/>
                                            <GradientStop Color="#FFADD1FF"
Offset="0.377"/>
```

```
                                        <GradientStop Color="#FFC0DBFF"
Offset="1"/>
                                    </LinearGradientBrush>
                                </Border.Background>
                            </Border>
                            <Border Name="brd" BorderThickness="0,0,1,0"
Margin="0,3"

BorderBrush="{StaticResource commandBaseBorderBrush}"/>
                            <ContentPresenter
                                  x:Name="contentPresenter"
                                  Content="{TemplateBinding Content}"
                                  ContentTemplate="{TemplateBinding
ContentTemplate}"
                                  HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
                                  VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
                                  Margin="{TemplateBinding Padding}"/>
                        </Grid>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
        <Style TargetType="RadioButton" x:Key="radioButtonStyle">
            <Setter Property="Width" Value="90"/>
            <Setter Property="Background" Value="{StaticResource
buttBackBrush}" />
            <Setter Property="Foreground" Value="#FF000000"/>
            <Setter Property="HorizontalContentAlignment" Value="Center"/>
            <Setter Property="VerticalContentAlignment" Value="Center"/>
            <Setter Property="FontSize" Value="14"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="RadioButton">
                        <Grid>
                            <VisualStateManager.VisualStateGroups>
                                <VisualStateGroup x:Name="CommonStates">
                                    <VisualState x:Name="Normal"/>
                                    <VisualState x:Name="MouseOver">
                                        <Storyboard>
                                            <DoubleAnimation Duration="0"
Storyboard.TargetName="brd" Storyboard.TargetProperty="Opacity" To="0.15"/>
                                            <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFFCDE"/>
                                            <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFEAAC"/>
                                            <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFD767"/>
```

```
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#FFFFE59B"/>
                                            </Storyboard>
                                        </VisualState>
                                    </VisualStateGroup>
                                    <VisualStateGroup x:Name="CheckStates">
                                        <VisualState x:Name="Checked">
                                            <Storyboard>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFD9AA"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFBB6E"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFAB3F"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#FFFEE17A"/>
                                            </Storyboard>
                                        </VisualState>
                                        <VisualState x:Name="Unchecked"/>
                                    </VisualStateGroup>
                                </VisualStateManager.VisualStateGroups>
                                <Border BorderThickness="0,0,1,0">
                                    <Border.Background>
                                        <LinearGradientBrush
x:Name="MouseOverGradient" StartPoint="0,0" EndPoint="0,1" >
                                            <GradientStop Color="Transparent"
Offset="0"/>
                                            <GradientStop Color="Transparent"
Offset="0.367"/>
                                            <GradientStop Color="Transparent"
Offset="0.377"/>
                                            <GradientStop Color="Transparent"
Offset="1"/>
                                        </LinearGradientBrush>
                                    </Border.Background>
                                </Border>
                                <Border Name="brd" BorderThickness="0,0,1,0">
                                    <Border.Background>
                                        <LinearGradientBrush
x:Name="BackgroundGradient" StartPoint="0,0" EndPoint="0,1" >
                                            <GradientStop Color="#FFE3EFFF"
Offset="0"/>
                                            <GradientStop Color="#F9C4DDFF"
Offset="0.367"/>
                                            <GradientStop Color="#E5ADD1FF"
Offset="0.377"/>
                                            <GradientStop Color="#C6C0DBFF"
```

```
Offset="1"/>
                                        </LinearGradientBrush>
                                    </Border.Background>
                                </Border>
                                <Border BorderThickness="0,0,1,0" Margin="0,3"

BorderBrush="{StaticResource commandBaseBorderBrush}"/>
                                <ContentPresenter
                                x:Name="contentPresenter"
                                Content="{TemplateBinding Content}"
                                ContentTemplate="{TemplateBinding
ContentTemplate}"
                                HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
                                VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
                                Margin="{TemplateBinding Padding}"/>
                            </Grid>
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </Style>
        </ResourceDictionary>
    </UserControl.Resources>
```

7. Add the following XAML markup to complete the styles and databindings used in your application:

XAML

```
<!-- Border Brush -->
            <LinearGradientBrush x:Key="BorderBrush" EndPoint="0.5,1"
StartPoint="0.5,0">
                <GradientStop Color="#FFA3AEB9" Offset="0"/>
                <GradientStop Color="#FF8399A9" Offset="0.375"/>
                <GradientStop Color="#FF718597" Offset="0.375"/>
                <GradientStop Color="#FF617584" Offset="1"/>
            </LinearGradientBrush>

            <!-- Button styles -->
            <SolidColorBrush x:Key="commandBaseBorderBrush"  Color="#FF6593CF"/>
            <LinearGradientBrush x:Key="buttBackBrush" StartPoint="0,0"
EndPoint="0,1" >
                <GradientStop Color="#FFE3EFFF" Offset="0"/>
                <GradientStop Color="#FFC4DDFF" Offset="0.367"/>
                <GradientStop Color="#FFADD1FF" Offset="0.377"/>
                <GradientStop Color="#FFC0DBFF" Offset="1"/>
            </LinearGradientBrush>
            <Style x:Key="buttonStyle"  TargetType="Button">
                <Setter Property="Width" Value="90"/>
                <Setter Property="Background" Value="{StaticResource
buttBackBrush}" />
                <Setter Property="Foreground" Value="#FF000000"/>
                <Setter Property="HorizontalContentAlignment" Value="Center"/>
```

```
                <Setter Property="VerticalContentAlignment" Value="Center"/>
                <Setter Property="FontSize" Value="14"/>
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate TargetType="Button">
                            <Grid>
                                <VisualStateManager.VisualStateGroups>
                                    <VisualStateGroup x:Name="CommonStates">
                                        <VisualState x:Name="Normal"/>
                                        <VisualState x:Name="MouseOver">
                                            <Storyboard>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFFCDE"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFEAAC"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFD767"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#FFFFE59B"/>
                                            </Storyboard>
                                        </VisualState>
                                        <VisualState x:Name="Pressed">
                                            <Storyboard>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFFCDE"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFEAAC"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFD767"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="Orange"/>
                                            </Storyboard>
                                        </VisualState>
                                    </VisualStateGroup>
                                </VisualStateManager.VisualStateGroups>
                                <Border BorderThickness="0,0,1,0">
                                    <Border.Background>
                                        <LinearGradientBrush
x:Name="BackgroundGradient" StartPoint="0,0" EndPoint="0,1" >
                                            <GradientStop Color="#FFE3EFFF"
Offset="0"/>
                                            <GradientStop Color="#FFC4DDFF"
Offset="0.367"/>
                                            <GradientStop Color="#FFADD1FF"
```

```
Offset="0.377"/>
                                        <GradientStop Color="#FFC0DBFF"
Offset="1"/>
                                    </LinearGradientBrush>
                                </Border.Background>
                            </Border>
                            <Border Name="brd" BorderThickness="0,0,1,0"
Margin="0,3"

BorderBrush="{StaticResource commandBaseBorderBrush}"/>
                            <ContentPresenter
                                  x:Name="contentPresenter"
                                  Content="{TemplateBinding Content}"
                                  ContentTemplate="{TemplateBinding
ContentTemplate}"
                                  HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
                                  VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
                                  Margin="{TemplateBinding Padding}"/>
                        </Grid>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
        <Style TargetType="RadioButton" x:Key="radioButtonStyle">
            <Setter Property="Width" Value="90"/>
            <Setter Property="Background" Value="{StaticResource
buttBackBrush}" />
            <Setter Property="Foreground" Value="#FF000000"/>
            <Setter Property="HorizontalContentAlignment" Value="Center"/>
            <Setter Property="VerticalContentAlignment" Value="Center"/>
            <Setter Property="FontSize" Value="14"/>
            <Setter Property="Template">
                <Setter.Value>
                    <ControlTemplate TargetType="RadioButton">
                        <Grid>
                            <VisualStateManager.VisualStateGroups>
                                <VisualStateGroup x:Name="CommonStates">
                                    <VisualState x:Name="Normal"/>
                                    <VisualState x:Name="MouseOver">
                                        <Storyboard>
                                            <DoubleAnimation Duration="0"
Storyboard.TargetName="brd" Storyboard.TargetProperty="Opacity" To="0.15"/>
                                            <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFFCDE"/>
                                            <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFEAAC"/>
                                            <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
```

```
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFD767"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="MouseOverGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#FFFFE59B"/>
                                            </Storyboard>
                                        </VisualState>
                                    </VisualStateGroup>
                                    <VisualStateGroup x:Name="CheckStates">
                                        <VisualState x:Name="Checked">
                                            <Storyboard>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[0].(GradientStop.Color)" To="#FFFFD9AA"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[1].(GradientStop.Color)" To="#FFFFBB6E"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[2].(GradientStop.Color)" To="#FFFFAB3F"/>
                                                <ColorAnimation Duration="0"
Storyboard.TargetName="BackgroundGradient" Storyboard.TargetProperty="
(GradientBrush.GradientStops)[3].(GradientStop.Color)" To="#FFFEE17A"/>
                                            </Storyboard>
                                        </VisualState>
                                        <VisualState x:Name="Unchecked"/>
                                    </VisualStateGroup>
                                </VisualStateManager.VisualStateGroups>
                                <Border BorderThickness="0,0,1,0">
                                    <Border.Background>
                                        <LinearGradientBrush
x:Name="MouseOverGradient" StartPoint="0,0" EndPoint="0,1" >
                                            <GradientStop Color="Transparent"
Offset="0"/>
                                            <GradientStop Color="Transparent"
Offset="0.367"/>
                                            <GradientStop Color="Transparent"
Offset="0.377"/>
                                            <GradientStop Color="Transparent"
Offset="1"/>
                                        </LinearGradientBrush>
                                    </Border.Background>
                                </Border>
                                <Border Name="brd" BorderThickness="0,0,1,0">
                                    <Border.Background>
                                        <LinearGradientBrush
x:Name="BackgroundGradient" StartPoint="0,0" EndPoint="0,1" >
                                            <GradientStop Color="#FFE3EFFF"
Offset="0"/>
                                            <GradientStop Color="#F9C4DDFF"
Offset="0.367"/>
                                            <GradientStop Color="#E5ADD1FF"
Offset="0.377"/>
```

```
                                        <GradientStop Color="#C6C0DBFF"
Offset="1"/>
                                    </LinearGradientBrush>
                                </Border.Background>
                            </Border>
                            <Border BorderThickness="0,0,1,0" Margin="0,3"

BorderBrush="{StaticResource commandBaseBorderBrush}"/>
                            <ContentPresenter
                            x:Name="contentPresenter"
                            Content="{TemplateBinding Content}"
                            ContentTemplate="{TemplateBinding
ContentTemplate}"
                            HorizontalAlignment="{TemplateBinding
HorizontalContentAlignment}"
                            VerticalAlignment="{TemplateBinding
VerticalContentAlignment}"
                            Margin="{TemplateBinding Padding}"/>
                        </Grid>
                    </ControlTemplate>
                </Setter.Value>
            </Setter>
        </Style>
    </ResourceDictionary>
</UserControl.Resources>
```

8. Insert the following XAML markup to create a grid, a border for your application, and a stackpanel that contains four radio buttons that will control the scheduler view:

XAML

```
<Grid x:Name="LayoutRoot" Background="White">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition/>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="Auto"/>
        </Grid.RowDefinitions>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="196"/>
            <ColumnDefinition MinWidth="200"/>
        </Grid.ColumnDefinitions>
        <Border BorderThickness="1" Height="28" HorizontalAlignment="Stretch"
Grid.Column="0" Grid.ColumnSpan="2"
                BorderBrush="{StaticResource BorderBrush}"
Background="{StaticResource buttBackBrush}">
            <StackPanel Orientation="Horizontal" >
                <RadioButton x:Name="btnDay" Style="{StaticResource
radioButtonStyle}" Content="Day"
                             c1:CommandExtensions.CommandTarget="{Binding
ElementName=Scheduler}"

c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
```

```
                                c1:CommandExtensions.CommandParameter="{Binding
Path=OneDayStyle, ElementName=Scheduler}"/>
                <RadioButton x:Name="btnWorkWeek" Style="{StaticResource
radioButtonStyle}" Content="Work Week"
                                c1:CommandExtensions.CommandTarget="{Binding
ElementName=Scheduler}"

c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
                                c1:CommandExtensions.CommandParameter="{Binding
Path=WorkingWeekStyle, ElementName=Scheduler}"/>
                <RadioButton x:Name="btnWeek" Style="{StaticResource
radioButtonStyle}" Content="Week"
                                c1:CommandExtensions.CommandTarget="{Binding
ElementName=Scheduler}"

c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
                                c1:CommandExtensions.CommandParameter="{Binding
Path=WeekStyle, ElementName=Scheduler}"/>
                <RadioButton x:Name="btnMonth" Style="{StaticResource
radioButtonStyle}" Content="Month"
                                c1:CommandExtensions.CommandTarget="{Binding
ElementName=Scheduler}"

c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
                                c1:CommandExtensions.CommandParameter="{Binding
Path=MonthStyle, ElementName=Scheduler}"/>
                <RadioButton x:Name="btnTimeLine" Style="{StaticResource
radioButtonStyle}" Content="Time Line" Width="100"
                                c1:CommandExtensions.CommandTarget="{Binding
ElementName=Scheduler}"

c1:CommandExtensions.Command="sched:C1Scheduler.ChangeStyleCommand"
                                c1:CommandExtensions.CommandParameter="{Binding
Path=TimeLineStyle, ElementName=Scheduler}"/>
            </StackPanel>
        </Border>
```

9. The following XAML markup creates the listbox, C1Calendar, and C1Scheduler for the application:

XAML

```
<c1sched:C1Calendar x:Name="Calendar" VerticalAlignment="Stretch" Grid.Column="0"
Grid.Row="1"
                                    MaxSelectionCount="42"
SelectedDates="{Binding VisibleDates, ElementName=Scheduler}"
                                    CalendarHelper="{Binding CalendarHelper,
ElementName=Scheduler}"
                                    BoldedDates="{Binding BoldedDates,
ElementName=Scheduler}"
                                    GenerateAdjacentMonthDays="true" Margin="2"/>
        <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, Mode=TwoWay}"/>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
        <c1sched:C1Scheduler x:Name="Scheduler"
StyleChanged="Scheduler_StyleChanged"
                                            GroupBy="Owner"
GroupHeaderTemplate="{StaticResource myCustomGroupHeaderTemplate}"
GroupPageSize="2"
                                            Grid.Column="1" Grid.Row="1"
Grid.RowSpan="2"
                                            Style="{Binding OneDayStyle,
ElementName=Scheduler}" c1:C1NagScreen.Nag="True">
            <c1sched:C1Scheduler.Settings>
                <c1sched:C1SchedulerSettings FirstVisibleTime="07:00:00"
AllowContactsEditing="False" AllowCategoriesEditing="False"
AllowCategoriesMultiSelection="False" />
            </c1sched:C1Scheduler.Settings>
        </c1sched:C1Scheduler>
        <!-- Status bar -->
        <StackPanel Margin="2" Grid.Row="3" Grid.ColumnSpan="2"
Orientation="Horizontal">
            <TextBlock Margin="2,2,10,2">Status:</TextBlock>
            <TextBlock Margin="2" x:Name="tbStatus" Grid.Row="2" MinHeight="10"/>
        </StackPanel>
    </Grid>
</UserControl>
```

In this step you created the application databindings and resources. You also created the C1Scheduler and C1Calendar controls and customized some of the scheduler settings. In the next step, you will add the code to control your application.

Step 3 of 4: Adding the Code to your Application

In this step you will add the code to control your application.

1. Right-click your application and select **View Code** from the list.
2. Add the following statements to the Import or using statements at the top of the page:

Visual Basic

```
Imports C1.Silverlight.Schedule
Imports System.Collections.Specialized
Imports C1.C1Schedule
Imports C1.Silverlight.Data
Imports System.IO
Imports System.Windows.Data
```

```
Imports System.Globalization
Imports MultiUser.DataService
```

C#

```
using C1.Silverlight.Schedule;
using System.Collections.Specialized;
using C1.C1Schedule;
using C1.Silverlight.Data;
using MultiUser.DataService;
using System.IO;
using System.Windows.Data;
using System.Globalization;
using System.ServiceModel;
```

3. Insert the following code below the initial class statement:

Visual Basic

```
Private _ds As DataSet = Nothing
Private _dtAppointments As DataTable = Nothing
Private _dtEmployees As DataTable = Nothing
Private _dtCustomers As DataTable = Nothing
```

C#

```
private DataSet _ds = null;
private DataTable _dtAppointments = null;
private DataTable _dtEmployees = null;
private DataTable _dtCustomers = null;
```

4. Add the following code directly underneath the **InitializeComponent()** method call to set C1Scheduler.ReminderFire and **Scheduler.GroupItems.CollectionChanged** event handlers:

Visual Basic

```
AddHandler Scheduler.ReminderFire, AddressOf scheduler_ReminderFire
AddHandler Scheduler.GroupItems.CollectionChanged, GroupItems_CollectionChanged
```

C#

```
Scheduler.ReminderFire += new EventHandler<ReminderActionEventArgs>
(scheduler_ReminderFire);
Scheduler.GroupItems.CollectionChanged += new
NotifyCollectionChangedEventHandler(GroupItems_CollectionChanged);
```

5. Set the mappings for the AppointmentStorage, OwnerStorage, and ContactStorage for the Scheduler:

Visual Basic

```
' set mappings for the AppointmentStorages
        Dim storage As AppointmentStorage =
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"

        ' set mappings for the OwnerStorage
        Dim ownerStorage As ContactStorage = Scheduler.DataStorage.OwnerStorage
        AddHandler DirectCast(ownerStorage.Contacts,
INotifyCollectionChanged).CollectionChanged, New
NotifyCollectionChangedEventHandler(AddressOf Owners_CollectionChanged)
        ownerStorage.Mappings.IndexMapping.MappingName = "EmployeeId"
        ownerStorage.Mappings.TextMapping.MappingName = "FirstName"

        ' set mappings for the ContactStorage
        Dim cntStorage As ContactStorage = Scheduler.DataStorage.ContactStorage
        AddHandler DirectCast(cntStorage.Contacts,
INotifyCollectionChanged).CollectionChanged, New
NotifyCollectionChangedEventHandler(AddressOf Contacts_CollectionChanged)
        cntStorage.Mappings.IdMapping.MappingName = "CustomerId"
        cntStorage.Mappings.TextMapping.MappingName = "CompanyName"

        ' load data from server
        LoadData()
```

C#

```
// set mappings for the AppointmentStorages
            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";

            // set mappings for the OwnerStorage
            ContactStorage ownerStorage = Scheduler.DataStorage.OwnerStorage;
            ((INotifyCollectionChanged)ownerStorage.Contacts).CollectionChanged
+= new NotifyCollectionChangedEventHandler(Owners_CollectionChanged);
            ownerStorage.Mappings.IndexMapping.MappingName = "EmployeeId";
            ownerStorage.Mappings.TextMapping.MappingName = "FirstName";

            // set mappings for the ContactStorage
            ContactStorage cntStorage = Scheduler.DataStorage.ContactStorage;
            ((INotifyCollectionChanged)cntStorage.Contacts).CollectionChanged +=
new NotifyCollectionChangedEventHandler(Contacts_CollectionChanged);
```

```
            cntStorage.Mappings.IdMapping.MappingName = "CustomerId";
            cntStorage.Mappings.TextMapping.MappingName = "CompanyName";

            // load data from server
            LoadData();
```

6. Add the following code to handle the **GroupItems_CollectionChanged** event:

Visual Basic

```
Private Sub GroupItems_CollectionChanged(sender As Object, e As
NotifyCollectionChangedEventArgs)
        For Each group As SchedulerGroupItem In Scheduler.GroupItems
            If group.Owner IsNot Nothing Then
                ' set SchedulerGroupItem.Tag property to the data row. That
allows to use data row fields in xaml for binding
                Dim index As Integer = CInt(group.Owner.Key(0))
                group.Tag = _dtEmployees.Rows.Find(index)
            End If
        Next
    End Sub
```

C#

```
void GroupItems_CollectionChanged(object sender,
NotifyCollectionChangedEventArgs e)
        {
            foreach (SchedulerGroupItem group in Scheduler.GroupItems)
            {
                if (group.Owner != null)
                {
                    // set SchedulerGroupItem.Tag property to the data row. That
allows to use data row fields in xaml for binding
                    int index = (int)group.Owner.Key[0];
                    group.Tag = _dtEmployees.Rows.Find(index);
                }
            }
        }
```

7. Use the following code to handle the **Owners_CollectionChanged** event:

Visual Basic

```
Private Sub Owners_CollectionChanged(sender As Object, e As
NotifyCollectionChangedEventArgs)
        If e.Action = NotifyCollectionChangedAction.Add Then
            For Each cnt As Contact In e.NewItems
                Dim row As DataRow = _dtEmployees.Rows.Find(cnt.Key(0))
                If row IsNot Nothing Then
                    ' set Contact.MenuCaption to the FirstName + LastName string
                    cnt.MenuCaption = row("FirstName").ToString() & " " &
row("LastName").ToString()
                End If
```

```
            Next
        End If
    End Sub
```

C#

```
void Owners_CollectionChanged(object sender, NotifyCollectionChangedEventArgs e)
        {
            if (e.Action == NotifyCollectionChangedAction.Add)
            {
                foreach (Contact cnt in e.NewItems)
                {
                    DataRow row = _dtEmployees.Rows.Find(cnt.Key[0]);
                    if (row != null)
                    {
                        // set Contact.MenuCaption to the FirstName + LastName
string
                        cnt.MenuCaption = row["FirstName"].ToString() + " " +
row["LastName"].ToString();
                    }
                }
            }
        }
```

8. Insert the code below to handle the **Contacts_CollectionChanged** event:

Visual Basic

```
Private Sub Contacts_CollectionChanged(sender As Object, e As
NotifyCollectionChangedEventArgs)
        If e.Action = NotifyCollectionChangedAction.Add Then
            For Each cnt As Contact In e.NewItems
                Dim row As DataRow = _dtCustomers.Rows.Find(cnt.Key(0))
                If row IsNot Nothing Then
                    ' set Contact.MenuCaption to the CompanyName + Contactname
string
                    cnt.MenuCaption = row("CompanyName").ToString() & " (" &
row("ContactName").ToString() & ")"
                End If
            Next
        End If
    End Sub
```

C#

```
void Contacts_CollectionChanged(object sender, NotifyCollectionChangedEventArgs
e)
        {
            if (e.Action == NotifyCollectionChangedAction.Add)
            {
                foreach (Contact cnt in e.NewItems)
                {
                    DataRow row = _dtCustomers.Rows.Find(cnt.Key[0]);
```

```
                    if (row != null)
                    {
                        // set Contact.MenuCaption to the CompanyName +
Contactname string
                        cnt.MenuCaption = row["CompanyName"].ToString() + " (" +
row["ContactName"].ToString() + ")";
                    }
                }
            }
        }
```

9. Add an event handler to prevent showing reminders:

Visual Basic

```
' prevent showing reminders

    Private Sub scheduler_ReminderFire(sender As Object, e As
ReminderActionEventArgs)

        e.Handled = True

    End Sub
```

C#

```
// prevent showing reminders

        private void scheduler_ReminderFire(object sender,
ReminderActionEventArgs e)

        {

            e.Handled = true;

        }
```

10. Insert the following region to handle loading data:

Visual Basic

```
#Region "** loading data"

    ' Get data service relative to current host/domain
    Private Function GetDataService() As DataServiceSoapClient

        ' Increase buffer size
        Dim binding = New System.ServiceModel.BasicHttpBinding()
        binding.MaxReceivedMessageSize = 2147483647

        ' int.MaxValue
        binding.MaxBufferSize = 2147483647
```

```
        ' int.MaxValue
        ' Get absolute service address

        Dim uri As Uri =
C1.Silverlight.Extensions.GetAbsoluteUri("DataService.asmx")
        Dim address = New System.ServiceModel.EndpointAddress(uri)

        ' Return new service client
        Return New DataServiceSoapClient(binding, address)
    End Function

    Private Sub LoadData()
        ' Invoke Web service
        Dim svc = GetDataService()
        AddHandler svc.GetDataCompleted, AddressOf svc_GetDataCompleted
        svc.GetDataAsync("Appointments,Employees,Customers")
    End Sub

    Private Sub svc_GetDataCompleted(sender As Object, e As
GetDataCompletedEventArgs)
        ' Handle errors
        If e.[Error] IsNot Nothing Then
            tbStatus.Text = "Error downloading data..."
            Return
        End If

        ' Parse data stream from server (DataSet as XML)
        tbStatus.Text = String.Format("Got data, {0:n0} kBytes", e.Result.Length
/ 1024)

        Dim ms = New MemoryStream(e.Result)
        _ds = New DataSet()
        _ds.EnforceConstraints = False
        _ds.ReadXml(ms)
        _ds.EnforceConstraints = True

        ' Got the data, find the table
        _dtAppointments = _ds.Tables("Appointments")
        _dtEmployees = _ds.Tables("Employees")
        _dtCustomers = _ds.Tables("Customers")

        ' set C1Scheduler data source to the DataTable loaded from server
        Scheduler.DataStorage.AppointmentStorage.DataSource = _dtAppointments
        Scheduler.DataStorage.ContactStorage.DataSource = _dtCustomers
        Scheduler.DataStorage.OwnerStorage.DataSource = _dtEmployees
    End Sub
#End Region
```

C#

```
#region ** loading data
        // Get data service relative to current host/domain
```

```
        DataServiceSoapClient GetDataService()
        {
            // Increase buffer size
            var binding = new System.ServiceModel.BasicHttpBinding();
            binding.MaxReceivedMessageSize = 2147483647; // int.MaxValue
            binding.MaxBufferSize = 2147483647; // int.MaxValue

            // Get absolute service address
            Uri uri =
C1.Silverlight.Extensions.GetAbsoluteUri("DataService.asmx");
            var address = new System.ServiceModel.EndpointAddress(uri);

            // Return new service client
            return new DataServiceSoapClient(binding, address);
        }

        void LoadData()
        {
            // Invoke Web service
            var svc = GetDataService();
            svc.GetDataCompleted += svc_GetDataCompleted;
            svc.GetDataAsync("Appointments,Employees,Customers");
        }

        void svc_GetDataCompleted(object sender, GetDataCompletedEventArgs e)
        {
            // Handle errors
            if (e.Error != null)
            {
                tbStatus.Text = "Error downloading data...";
                return;
            }

            // Parse data stream from server (DataSet as XML)
            tbStatus.Text = string.Format(
              "Got data, {0:n0} kBytes", e.Result.Length / 1024);

            var ms = new MemoryStream(e.Result);
            _ds = new DataSet();
            _ds.EnforceConstraints = false;
            _ds.ReadXml(ms);
            _ds.EnforceConstraints = true;

            // Got the data, find the table
            _dtAppointments = _ds.Tables["Appointments"];
            _dtEmployees = _ds.Tables["Employees"];
            _dtCustomers = _ds.Tables["Customers"];

            // set C1Scheduler data source to the DataTable loaded from server
            Scheduler.DataStorage.AppointmentStorage.DataSource =
_dtAppointments;
```

```
            Scheduler.DataStorage.ContactStorage.DataSource = _dtCustomers;
            Scheduler.DataStorage.OwnerStorage.DataSource = _dtEmployees;
        }
        #endregion
```

11. The following code handles the **Scheduler_StyleChanged** event:

Visual Basic

```
Private Sub Scheduler_StyleChanged(sender As Object, e As RoutedEventArgs)
        If Scheduler.Style Is Scheduler.TimeLineStyle Then
            ' update group header (use different headers for TimeLine and other
views)
            Scheduler.GroupHeaderTemplate =
DirectCast(Resources("myCustomTimeLineGroupHeaderTemplate"), DataTemplate)
            btnTimeLine.IsChecked = True
        Else
            ' update group header (use different headers for TimeLine and other
views)
            Scheduler.GroupHeaderTemplate =
DirectCast(Resources("myCustomGroupHeaderTemplate"), DataTemplate)
            ' update toolbar buttons state according to the current C1Scheduler
view
            If Scheduler.Style Is Scheduler.WorkingWeekStyle Then
                btnWorkWeek.IsChecked = True
            ElseIf Scheduler.Style Is Scheduler.WeekStyle Then
                btnWeek.IsChecked = True
            ElseIf Scheduler.Style Is Scheduler.MonthStyle Then
                btnMonth.IsChecked = True
            Else
                btnDay.IsChecked = True
            End If
        End If
    End Sub
```

C#

```
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
            {
                // update group header (use different headers for TimeLine and
other views)
                Scheduler.GroupHeaderTemplate =
```

```
(DataTemplate)Resources["myCustomGroupHeaderTemplate"];
                // update toolbar buttons state according to the current
C1Scheduler view
                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;
                }
            }
        }
```

12. The last section of code to insert is the code that creates the **BooleanToVisibilityConverter**:

Visual Basic

```
Public Class BooleanToVisibilityConverter
        Implements IValueConverter
        Public Function Convert(value As Object, targetType As Type, parameter
As Object, culture As CultureInfo) As Object
            If TypeOf value Is [Boolean] Then
                Return If(CBool(value), Visibility.Visible,
Visibility.Collapsed)
            End If
            Return value
        End Function
        Public Function ConvertBack(value As Object, targetType As Type,
parameter As Object, culture As CultureInfo) As Object
            Throw New NotImplementedException()
        End Function
        Public Function Convert1(value As Object, targetType As System.Type,
parameter As Object, culture As System.Globalization.CultureInfo) As Object
Implements System.Windows.Data.IValueConverter.Convert
            Return value
        End Function
        Public Function ConvertBack1(value As Object, targetType As System.Type,
parameter As Object, culture As System.Globalization.CultureInfo) As Object
Implements System.Windows.Data.IValueConverter.ConvertBack
            Return value
        End Function
    End Class
End Class
```

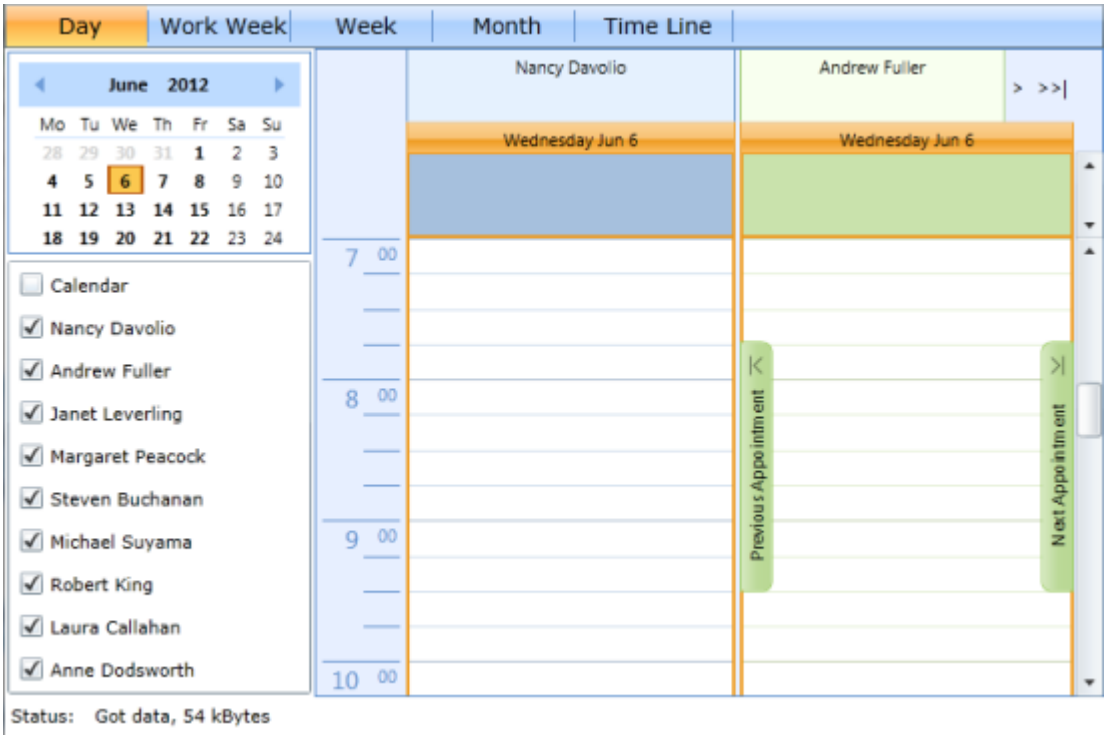
C#

```
public class BooleanToVisibilityConverter : IValueConverter
        {
            public object Convert(object value, Type targetType, object
parameter,
                                  CultureInfo culture)
            {
                if (value is Boolean)
                {
                    return ((bool)value) ? Visibility.Visible :
Visibility.Collapsed;
                }
                return value;
            }
            public object ConvertBack(object value, Type targetType, object
parameter,
                                      CultureInfo culture)
            {
                throw new NotImplementedException();
            }
        }
    }
}
```

Step 4 of 4: Running the Application

In this step you will run the application.

Press **F5** to run your application. The application should appear similar to the following image:



Note that you can control the users displayed in the listbox at the left. You can also add an appointment to any user's schedule by double-clicking on the day. You can also control the view with the buttons at the top of the application.

What You've Accomplished

You created a new Silverlight application and added code files and a Data Service. You also added an Access database as a resource for your application. You used code and XAML markup to create and control the design and function of your application.

C1Scheduler Samples

Please be advised that this software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with ComponentOne Studio.

Please refer to the pre-installed product samples through the following path:

Documents | ComponentOne Samples | WPF, or

Documents | ComponentOne Samples | Silverlight

The following sample pages details the C1Scheduler control:

WPF

C# Samples

Sample	Description
Themes and Views	Demonstrates how to use the themes, views, time scales, and work times for C1Scheduler.
Grouping	Shows how to group the C1Scheduler control by resources, contacts, and categories.
Custom Styles	Shows how to create custom styles for the C1Scheduler control using custom defined styles instead of predefined C1Scheduler views.

The following samples can be accessed from the **C1WPFSchedule Samples**. To view samples, on your desktop, click the **Start** button and then click **ComponentOne | Studio for WPF | Samples | C1WPFSchedule Samples**.

Sample	**Description**
BusinessObjectsBinding	Shows how to bind AppointmentStorage to the collection of a custom business object.
CustomDialogs	Demonstrates replacing default C1Scheduler dialogs by custom ones.
CustomGroupingView	Demonstrates how to use custom-defined Day/Working Week/ Week C1Scheduler grouping styles instead of predefined ones.
CustomUIElements	Demonstrates customizing of a Time Ruler and IntervalAppointmentTemplate.
Grouping	Shows C1Scheduler grouping functionality. Uses Calendar for navigation. Change grouping criteria or navigate to different groups.
Localization	Demonstrates creating and using localized string resources.
Multiuser	Demonstrates showing multi-user calendar. AppoinmentStorage, OwnerStorage and ContactStorage are bound to data from NWind database Other storages are used in unbound mode. The list of all users is shown in the left bottom corner. C1Scheduler only shows groups for the checked users. "Calendar" group shows appointments with an empty owner. End-user can drag appointments from the one calendar to another. In such

	case Appointment.Owner property gets changed.
PrintDocumentTemplates	Demonstrates creating C1.C1Preview.C1PrintDocument templates for printing C1Scheduler's views.
Printing	Demonstrates printing C1Scheduler's views via the C1.C1Preview.C1PrintDocument component.

Silverlight

C# Samples

Sample	Description
Default Views	Demonstrates the four built-in views: day, week, work week, and month.
Navigation	Demonstrates switching **C1Scheduler**'s views and navigation through the control using the **C1Calendar** control.
ClearStyle	Demonstrates ComponentOne ClearStyle technology that allows you to create a custom style for the **C1Scheduler** control without having to hassle with XAML templates and style resources.
Custom Views	Demonstrates the use of custom-defined styles instead of predefined **C1Scheduler** views.

Other C# Samples shipped with the C1Scheduler control:

Sample	**Description**
AdvancedIntervalAppointment	Demonstrates advanced custom IntervalAppointmentTemplate scenario.
BusinessObjectsBinding	Demonstrates binding C1Scheduler's AppointmentStorage to collection of custom business objects.
CustomDialogs	Demonstrates replacing default C1Scheduler dialogs by custom ones.
CustomGroupingView	Demonstrates using of custom-defined Day/Working Week/Week C1Scheduler grouping styles instead of predefined ones.
DataBinding	Shows how to implement data-bound scheduling application.
Grouping	Demonstrates C1Scheduler grouping functionality.
IntervalAppointmentTemplate	Demonstrates using custom IntervalAppointmentTemplate.
Localization	Demonstrates localization of C1Scheduler.
MultiUser	Demonstrates showing multi-user calendar.
Theming	Demonstrates using C1Scheduler with C1.Silverlight.Theming.

Calendar Samples

Please be advised that this software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with ComponentOne Studio.

Please refer to the pre-installed product samples through the following path:

Documents | ComponentOne Samples | WPF, or

Documents | ComponentOne Samples | Silverlight

The following sample pages details the C1Calendar control:

WPF

C# Samples

Sample	Description
Calendar Settings	Shows how to use the calendar settings to change week start, work days, add holidays or working weekends.
Calendar	Displays the various appearance properties for the C1Calendar control.

Silverlight

C# Samples

Sample	Description
Calendar	Displays the various appearance properties for the C1Calendar control.
Navigation	Demonstrates switching **C1Scheduler**'s views and navigation through the control using the **C1Calendar** control.

C1Scheduler Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio and Microsoft Blend. By following the steps outlined in the Help, you will be able to create projects demonstrating a variety of Scheduler for Silverlight features and get a good sense of what Scheduler for Silverlight can do.

Each task-based help topic also assumes that you have created a new .NET or Blend project with a reference to the **C1.WPF.C1Schedule** or **C1.Silverlight.C1Schedule** assembly.

C1Scheduler Tasks

The following topics show how to perform specific C1Scheduler tasks.

Using the Edit Template to Create a Copy of C1Scheduler

To create and use a copy of the C1Scheduler source xaml using Expression Blend:

1. Select C1Scheduler instance on designer surface.
2. Select **Edit Template | Edit a Copy** from the Blend context menu.
3. Check style key (default value is 'C1SchedulerStyle1') and xaml placement in the "Create Style Resource" dialog and press **Ok** button.
4. Review Blend-generated xaml. It contains exact copy of all C1Scheduler styles and resources. There are 5 styles which can be used in the C1Scheduler: *OneDayStyle*, *WeekStyle*, *WorkingWeekStyle*, *MonthStyle* and *TimeLineStyle*.
5. Use the default placeholder style (due to limitations) which is currently applied to the C1Scheduler.
6. Remove the next.xaml generated by Blend:

   Style="{StaticResource C1SchedulerStyle1}".

7. Decide what styles you will customize and use them to set C1Scheduler Style, OneDayStyle, WeekStyle, WorkingWeekStyle, MonthStyle or TimeLineStyle properties. For example: MonthStyle="{StaticResource MonthStyle}".
8. Build and run your application to review the results.

Now you can customize styles and resources according your needs.

Setting Mappings and DataSource for the ContactStorage

To set the mappings and datasource for the ContactStorage you can use the following code:

C#

```
// set mappings and DataSource for the ContactStorage
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;
```

Linking a Scheduler to a Calendar

The following topic explains how to bind a schedule to a C1Calendar control in Microsoft Blend, Visual Studio, and using XAML.

Using Microsoft Blend

In Blend, linking C1Scheduler to one of the calendar controls is as easy as setting a property.

1. Add a C1Scheduler and C1Calendar control to your window.
2. Under **Objects and Timeline**, click **scheduler1**, and enter **C1Scheduler1** as the name of the control. If necessary, select **Window | Interaction** to view **Objects and Timeline**.
3. Select the C1Calendar control you added to the window.
4. In the **Properties** panel of **Design** view, click the **Advanced Property Options** button next to the **C1Calendar.SelectedDate** property in the **DateTime** category.
5. Select **Custom Expression** and enter the following expression:

   {Binding Path=SelectedDateTime, ElementName=C1Scheduler1, Mode=TwoWay}

   This will bind the selected date of the calendar to the schedule.

   The equivalent XAML code can be viewed by clicking the **XAML** view tab. It should look like the following:

   XAML

   ```
   c1:C1Calendar HorizontalAlignment="Left" VerticalAlignment="Top" Theme="
   {DynamicResource {ComponentResourceKey ResourceId=Default, TypeInTargetAssembly=
   {x:Type c1sched:CalendarBase}}}" SelectedDate="{Binding SelectedDateTime,
   ElementName=C1Scheduler1, Mode=TwoWay}">
   ```

6. Press **F5** to run the project and select a date in the calendar. The schedules selected date will change accordingly.

Using Visual Studio

To link C1Scheduler to a C1Calendar control:

1. Add a C1Scheduler and C1Calendar control to your window.
2. Select the C1Scheduler control.
3. In the Properties window, enter **C1Scheduler1** in the **Name** text box, if necessary.
4. Select the C1Calendar control you added to the window.
5. In the XAML window, find the <my:C1Calendar /> XAML.
6. Edit the XAML so it looks similar the following:
```
<c1:C1Calendar HorizontalAlignment="Left" Margin="34,51,0,0" Name="c1Calendar1"
VerticalAlignment="Top" SelectedDate="{Binding Path=SelectedDateTime,
ElementName=C1Scheduler1, Mode=TwoWay}"  />
```
7. Press **F5** to run the project and select a date in the C1Calendar. The schedule's selected date will change accordingly.

Using XAML

The following XAML binds C1Scheduler to a C1Calendar control:

```
<c1:C1Calendar HorizontalAlignment="Left" Margin="34,51,0,0" Name="c1Calendar1"
VerticalAlignment="Top" SelectedDate="{Binding Path=SelectedDateTime,
ElementName=C1Scheduler1, Mode=TwoWay}"  />
```

Customizing the Time Column

To customize the time column of a C1Scheduler control, you can edit the built-in *C1Scheduler_TimeRuler_Template*. In the following examples, we will edit the **TimeRuler** datatemplate to change the way the time appears in the time column and to change the background color of the time column.

Changing the layout of the time column

1. In the Visual Studio Solution Explorer, right-click the project name and select **Add > Resource Dictionary**.
2. Name your resource dictionary and click **Add**.
3. Add the C1Scheduler namespace so it looks like the following:

XAML
```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:c1="clr-
namespace:C1.Silverlight.C1Schedule;assembly=C1.Silverlight.C1Schedule"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
```

4. Find and open **Default.xml** installed with the product to **C:\Program Files\ComponentOne\WPF Edition\C1Schedule\XAML\themes\SchedulerThemes\Office2007** or **C:\Program Files\ComponentOne\Silverlight Edition\C1Schedule\XAML\themes\SchedulerThemes\Office2007**.
5. Find the datatemplate with the **C1Scheduler_TimeRuler_Template** key.

XAML
```
<!-- determines the template used for one hour of a time ruler in a Day view -->
      <DataTemplate x:Key="C1Scheduler_TimeRuler_Template">
            <Grid Name="OneHourGrid">
                  <Grid.ColumnDefinitions>
                        <ColumnDefinition Width="Auto" />
                        <ColumnDefinition MinWidth="20"
SharedSizeGroup="Minutes" />
```

```
                </Grid.ColumnDefinitions>
                <ItemsControl Grid.Column="1" x:Name="IntervalMarkersList"
ItemsSource="{Binding Items}">
                    <ItemsControl.ItemTemplate>
                        <DataTemplate>
                            <Border BorderThickness="0,1px,0,0"
Margin="0,0,4,0" SnapsToDevicePixels="True"
                                    BorderBrush="{DynamicResource
C1Scheduler_ControlAreaLines_Brush}" />
                        </DataTemplate>
                    </ItemsControl.ItemTemplate>
                    <ItemsControl.ItemsPanel>
                        <ItemsPanelTemplate>
                            <UniformGrid Rows="{Binding
ElementName=IntervalMarkersList, Path=Items.Count}" />
                        </ItemsPanelTemplate>
                    </ItemsControl.ItemsPanel>
                </ItemsControl>
                <Border BorderThickness="0,1px,0,0" Margin="4,0,0,0"
Grid.Column="0" SnapsToDevicePixels="True"
                        BorderBrush="{DynamicResource
C1Scheduler_ControlAreaLines_Brush}"
                        HorizontalAlignment="Right"
VerticalAlignment="Top" MinWidth="25">
                    <TextBlock
                            FontSize="16" Foreground="{DynamicResource
C1Scheduler_ControlAreaText_Brush}"
                            HorizontalAlignment="Right"
Padding="3,0,3,0" >
                        <TextBlock.Text>
                            <MultiBinding Converter="{x:Static
c1sched:DateTimeInfoToStringConverter.Default}">
                                <Binding Path="StartTimeInfo"/>
                                <Binding Source="hh" />
                            </MultiBinding>
                        </TextBlock.Text>
                    </TextBlock>
                </Border>
                <TextBlock FontSize="11" Grid.Column="1"
                        Foreground="{DynamicResource
C1Scheduler_ControlAreaText_Brush}"
                        HorizontalAlignment="Right" Padding="3,2,2,0"
Margin="0,0,4,0">
                        <TextBlock.Text>
                            <Binding Converter="{x:Static
c1sched:TimeRulerConverter.Default}"

Path="VisualIntervals[0].StartTimeInfo" />
                        </TextBlock.Text>
                </TextBlock>
            </Grid>
```

```
    </DataTemplate>
```

6. Copy and paste the datatemplate into your resource dictionary.
7. Change the following XAML:

```
<TextBlock Text="{Binding
VisualIntervals[0].StartTimeInfo.FormattedDate[hh]}"
```

So that it looks like this:

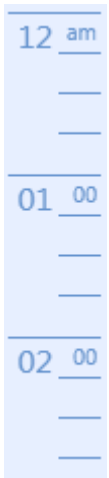
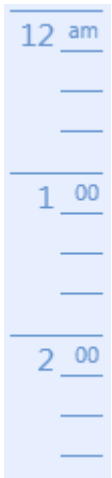
```
<TextBlock Text="{Binding
VisualIntervals[0].StartTimeInfo.FormattedDate[%h]}"
```

8. In the Solution Explorer, double-click Window1.xaml or the main window/page of your project.
9. Add a resource to point to your resource dictionary:

XAML

```
<Window.Resources>
<ResourceDictionary>
<ResourceDictionary x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type
my:C1Scheduler},
ResourceId=custom_theme}" Source="MyResourceDictionary.xaml" />
</ResourceDictionary>
</Window.Resources>
```

Here is an example of what the time column looked like before changing the datatemplate as well as what it looks like once the custom format string is changed:

hh Format String	%h Format String
	

Changing the color of the time column

1. In the Visual Studio Solution Explorer, right-click the project name and select **Add > Resource Dictionary**.
2. Name your resource dictionary and click **Add**.
3. Add the C1Scheduler namespace so it looks like the following:

XAML

```
<ResourceDictionary
```

```
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:c1="clr-
namespace:C1.Silverlight.C1Schedule;assembly=C1.Silverlight.C1Schedule"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
```

4. Find and open **Default.xml** installed with the product to **C:\Program Files\ComponentOne\WPF Edition\C1Schedule\XAML\themes\SchedulerThemes\Office2007** or **C:\Program Files\ComponentOne\Silverlight Edition\C1Schedule\XAML\themes\SchedulerThemes\Office2007**.
5. Find the datatemplate with the **C1Scheduler_TimeRuler_Template** key.

XAML

```
<!-- determines the template used for one hour of a time ruler in a Day view -->
<DataTemplate x:Key="C1Scheduler_TimeRuler_Template">
<Grid Name="OneHourGrid">
<Grid.ColumnDefinitions>
<ColumnDefinition Width="Auto" />
<ColumnDefinition MinWidth="20" SharedSizeGroup="Minutes" />
</Grid.ColumnDefinitions>
<ItemsControl Grid.Column="1" x:Name="IntervalMarkersList" ItemsSource="{Binding
Items}">
<ItemsControl.ItemTemplate>
<DataTemplate>
<Border BorderThickness="0,1px,0,0" Margin="0,0,4,0" SnapsToDevicePixels="True"
BorderBrush="{DynamicResource C1Scheduler_ControlAreaLines_Brush}" />
</DataTemplate>
</ItemsControl.ItemTemplate>
<ItemsControl.ItemsPanel>
<ItemsPanelTemplate>
<UniformGrid Rows="{Binding ElementName=IntervalMarkersList, Path=Items.Count}"
/>
</ItemsPanelTemplate>
</ItemsControl.ItemsPanel>
</ItemsControl>
<Border BorderThickness="0,1px,0,0" Margin="4,0,0,0" Grid.Column="0"
SnapsToDevicePixels="True"
BorderBrush="{DynamicResource C1Scheduler_ControlAreaLines_Brush}"
HorizontalAlignment="Right" VerticalAlignment="Top" MinWidth="25">
<TextBlock Text="{Binding VisualIntervals[0].StartTimeInfo.FormattedDate[hh]}"
FontSize="16"
Foreground="{DynamicResource C1Scheduler_ControlAreaText_Brush}"
HorizontalAlignment="Right" Padding="3,0,3,0" />
</Border>
<TextBlock FontSize="11" Grid.Column="1"
Foreground="{DynamicResource C1Scheduler_ControlAreaText_Brush}"
HorizontalAlignment="Right" Padding="3,2,2,0" Margin="0,0,4,0">
<TextBlock.Text>
<Binding Converter="{x:Static c1sched:TimeRulerConverter.Default}"
Mode="OneWay" Path="VisualIntervals[0].StartTimeInfo" />
</TextBlock.Text>
</TextBlock>
</Grid>
```

```
</DataTemplate>
```

6. Copy and paste the datatemplate into your resource dictionary.
7. Edit the **Grid Name** to include a background color:

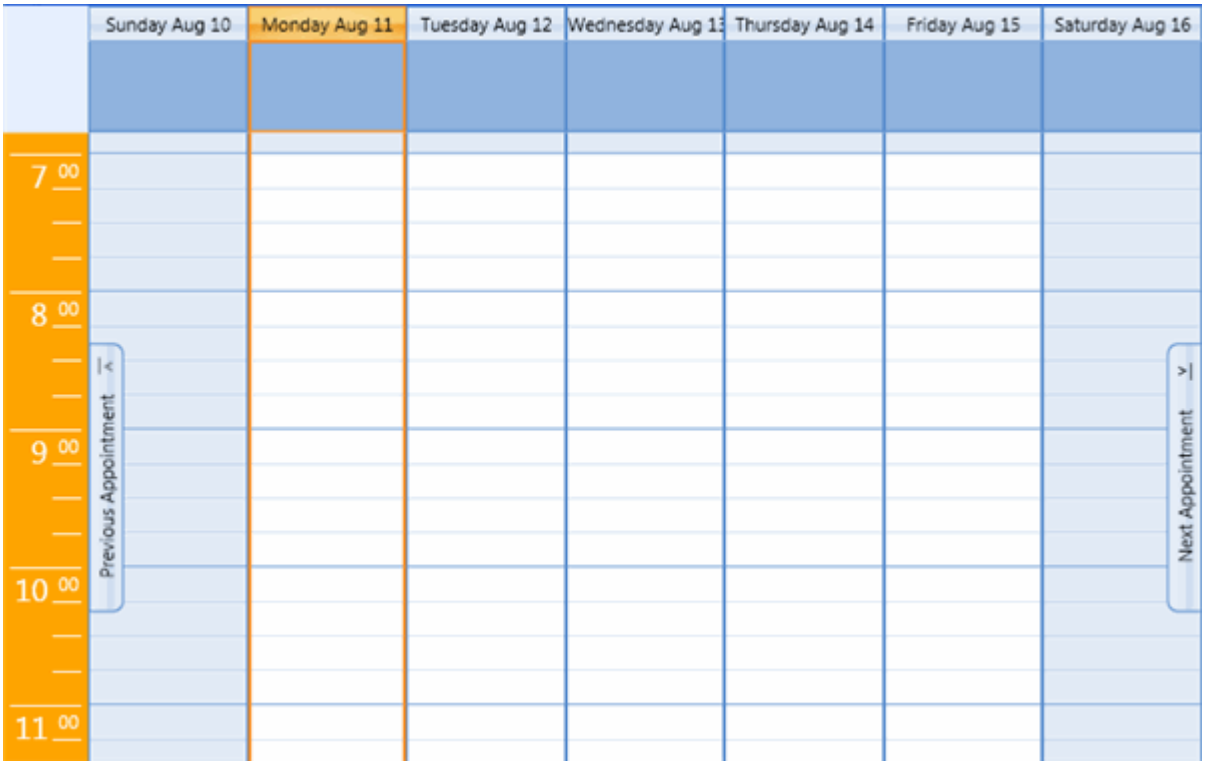
```
<Grid Name="OneHourGrid" Background="Orange">
```

8. In the Solution Explorer, double-click Window1.xaml or the main window/page of your project.
9. Add a resource to point to your resource dictionary:

XAML

```
<Window.Resources>
<ResourceDictionary>
<ResourceDictionary x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type
my:C1Scheduler},
ResourceId=custom_theme}" Source="MyResourceDictionary.xaml" />
</ResourceDictionary>
</Window.Resources>
```

This XAML creates a schedule that looks like the following image:



Customizing Time Span for Different Views

Scheduler for WPF and Silverlight enables users to customize the time span for views with visual intervals shorter than a day that includes Day, WorkWeek, Week, and TimeLine views. The C1Scheduler class provides the SmallVisualIntervalScale property (a dependency property as its depends upon VisualIntervalScale property) to set the time span in XAML.

You can set the SmallVisualIntervalScale property to a particular value or bind it with a scheduler element, such as

scale, to update the time span whenever the view is changed by the end-user. By default, the **SmallVisualIntervalScale** property is set to TimeSpan.Zero. Due to this, only VisualScaleInterval property has effect unless SmallVisualIntervalScale property is not set to a value or bound to some element in XAML.

Developers can use this property to apply and preserve custom scaling while switching from one View to another.

The following XAML code snippet shows how to bind SmallVisualIntervalScale property to scale element.

XAML

```
<c1sched:C1Scheduler x:Name="scheduler1" Grid.Column="1" Margin="10 0 0 0"
BorderThickness="1"
ShowWorkTimeOnly="True" ViewType="Month"  SmallVisualIntervalScale="{Binding
SelectedItem, ElementName=scale}">
```

Use the following C# code to initialize the scale element in the interaction logic for XAML. The SmallVisualIntervalScale property used in the above code example binds to the scale element initialized in the code given below.

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));
```

Changing Navigation Pane Text

To change the navigation pane text, simply set the C1Scheduler.NextAppointmentText and C1Scheduler.PreviousAppointmentText properties.

Using Microsoft Blend

To change the navigation pane text in Blend:

1. Add a C1Scheduler control to your window.
2. In the **Properties** panel of **Design** view, expand the **Appearance** node, and enter **Forward** for the C1Scheduler.NextAppointmentText property.
3. Enter **Back** for the C1Scheduler.NextAppointmentText property.

Using Visual Studio

To change the navigation pane text in Visual Studio:

1. Add a C1Scheduler control to your window.
2. In the Properties window, set the C1Scheduler.NextAppointmentText property to **Forward**.
3. Set the C1Scheduler.PreviousAppointmentText property to **Back**.

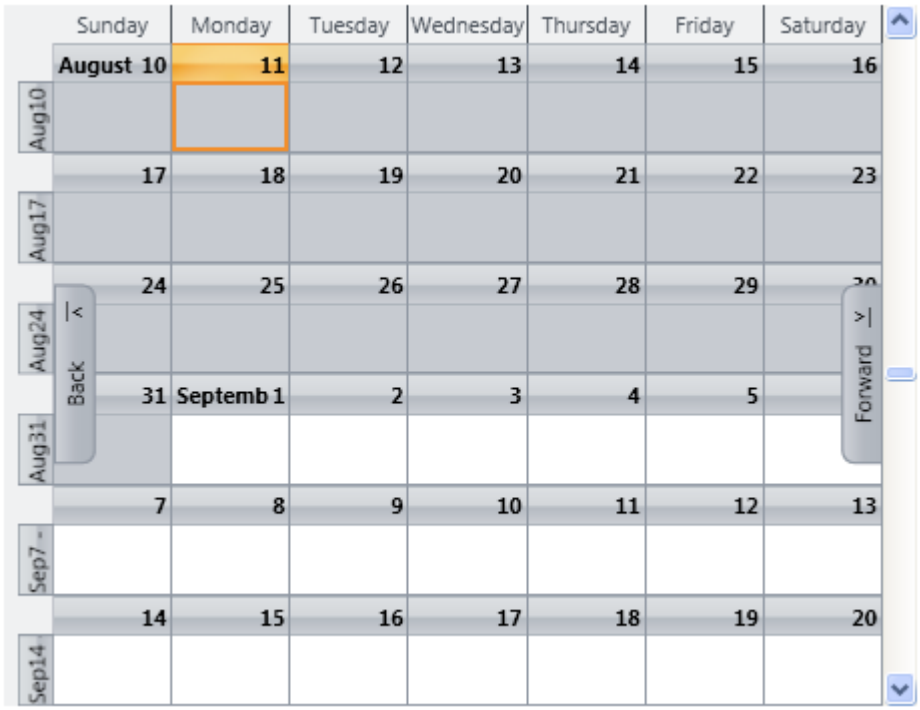
Using XAML

The following XAML sets the C1Scheduler.NextAppointmentText and C1Scheduler.PreviousAppointmentText properties.

XAML

```
<c1:C1Scheduler Margin="15,15,0,12" Name="C1Scheduler2" Grid.ColumnSpan="2"
NextAppointmentText="Forward" PreviousAppointmentText="Back" Theme="{DynamicResource
{ComponentResourceKey TypeInTargetAssembly=c1sched:C1Scheduler,
ResourceId=Office2007.Default}}"></c1:C1Scheduler>
```

The schedule will now look similar to the following image:



Setting Mappings and DataSource for the ContactStorage

To set the mappings and datasource for the ContactStorage you can use the following code:

C#

```
// set mappings and DataSource for the ContactStorage
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;
```

Setting the Days for Working Week View

To specify the days of the week to appear in **Working Week View**, you can set the WorkDays through the C1Scheduler.CalendarHelper property.

Using Blend

To set the days for **Working Week View** in Blend:

1. Add a C1Scheduler control to your window and select it.
2. In the **Properties** panel of **Design** view, expand the **Appearance** node.
3. Click the drop-down arrow next to **Style**, and select **Working Week View**.
4. Expand the **Calendar** node and then expand the **CalendarHelper** node.
5. Click the drop-down arrow next to **WorkDays** and check the checkbox next to each day you want to appear in the working week.

Using Visual Studio

To set the days for **Working Week View** in Visual Studio:

1. Add a C1Scheduler control to your window.
2. In the Properties window, set the **Style** property to **Working Week View**.
3. Expand the C1Scheduler.CalendarHelper property and click the drop-down arrow next to **WorkDays**.
4. Check the checkbox next to each day you want to appear in the working week.

Using XAML

The following XAML sets the days of the week for **Working Week View**:

XAML

```
<c1:C1Scheduler x:Name="scheduler1" Theme="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=Office2007.Default}}"
Style="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=WorkingWeekStyle}}">
       <c1:C1Scheduler.CalendarHelper>
             <c1:CalendarHelper Culture="English " WeekStart="Sunday"
                    EndDayTime="18:20:00" StartDayTime="09:20:00"
                    WorkDays="Tuesday,Wednesday,Thursday,Friday,Saturday">
             </c1:CalendarHelper>
       </c1:C1Scheduler.CalendarHelper>
</c1:C1Scheduler>
```

Adding Holidays to the C1Scheduler

You can add holidays to the C1Scheduler control using the CalendarHelper.Holidays property. This property can not be set in XAML; it can only be set in code.

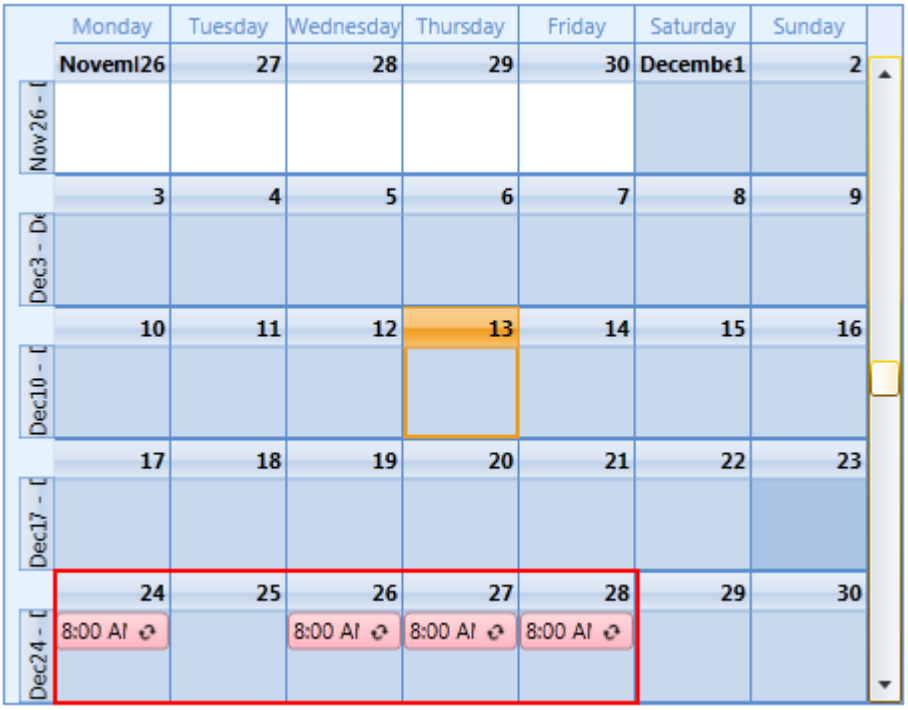
Use the following code to add holidays to the C1Scheduler:

XAML

```
sched1.CalendarHelper.Holidays.Add(new DateTime(2013, 1, 1));
sched1.CalendarHelper.Holidays.Add(new DateTime(2012, 12, 25));
sched1.CalendarHelper.Holidays.Add(new DateTime(2012, 12, 31));
```

This topic illustrates the following:

The C1Scheduler control will not allow the user to create an appointment on a day designated as a holiday:



Grouping

You can easily start grouping by using the C1Scheduler.GroupBy property, which determines the type of grouping to display. The C1Scheduler control supports grouping by contacts, categories, resources, and by the Appointment.Owner property value.

String	Grouping
Empty string	No grouping.
Category	Grouping is determined by the Appointment.Categories property value.
Contact	Grouping is determined by the Appointment.Links property value.
Owner	Grouping is determined by the Appointment.Owner property value.
Resource	Grouping is determined by the Appointment.Resources property value.

Suppose you have a schedule linked to a calendar in your project. Your XAML might look similar to the following:

XAML (WPF)

```
<Window x:Class="Grouping.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="574" Width="757"
        xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml"
        xmlns:c1sched="http://schemas.componentone.com/wpf/Schedule"
Loaded="Window_Loaded">
   <Grid Height="371" Width="689">
```

```
    <c1:C1Calendar x:Name="cal1" CalendarHelper="{Binding CalendarHelper,
ElementName=sched1, Mode=OneWay}"
      SelectedDates="{Binding VisibleDates, ElementName=sched1, Mode=OneWay}"
       BoldedDates="{Binding BoldedDates, ElementName=sched1, Mode=OneWay}"
      MaxSelectionCount="42" Margin="-9,7,459,188" />
    <c1:C1Scheduler x:Name="sched1" Margin="195,0,12,62"/>
   </Grid>
</Window>
```

XAML (Silverlight)

```
<UserControl x:Class="SilverlightApplication11.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="540" d:DesignWidth="687"
xmlns:c1="http://schemas.componentone.com/winfx/2006/xaml">

    <Grid x:Name="LayoutRoot" Background="White" Height="434" Width="591">
        <c1:C1Calendar x:Name="cal1" CalendarHelper="{Binding CalendarHelper,
ElementName=sched1, Mode=OneWay}"
      SelectedDates="{Binding VisibleDates, ElementName=sched1, Mode=OneWay}"
      BoldedDates="{Binding BoldedDates, ElementName=sched1, Mode=OneWay}"
      MaxSelectionCount="42" Margin="-9,7,459,188" />
        <c1:C1Scheduler x:Name="sched1" Margin="195,0,12,62"/>

    </Grid>
</UserControl>
```

Double-click the **MainWindow** and add the following code to the **Window_Loaded** event:

Visual Basic

```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
      sched1.GroupBy = "Category"
End Sub
```

C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            sched1.GroupBy = "Category";
        }
```

When you run the project, the schedule will be grouped by categories. This is a simplified example. For a full sample, see the **Grouping** sample that is installed with the product.

C1Calendar Tasks

The following topics show how to perform specific calendar tasks.

Changing the Calendar Month or Year

You can easily change the selected month or year of a C1Calendar control using the CalendarBase.SelectedDate property.

Using XAML

The following XAML sets the CalendarBase.SelectedDate property:

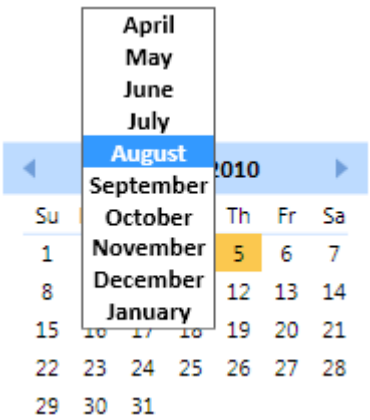
XAML

```
<c1:C1Calendar Margin="10,10,82,89" Name="c1Calendar1" HorizontalAlignment="Left"
Width="182" Height="159" VerticalAlignment="Top" SelectedDate="2010-08-5" />
```

At Run Time

To change the month or year of a C1Calendar control at run time:

1. Click the month or year at the top of the calendar. A pop-up list appears.



2. Select the desired month or year. If the month or year you are looking for does not appear in the list, select the last month or year and click the month or year again to see a new list.

Setting the Maximum and Minimum Allowable Dates

To set the maximum and minimum allowable dates for the C1Calendar, use the CalendarBase.MaxDate and CalendarBase.MinDate properties.

Using Visual Studio

To set the maximum and minimum allowable dates for the C1Calendar in Visual Studio:

1. Add a C1Calendar control to your window and select it.
2. In the Properties window, enter '12/31/2099' next to the CalendarBase.MaxDate property.
3. In the Properties window, enter '1/1/1700' next to the CalendarBase.MinDate property.

Using XAML

The following XAML sets the CalendarBase.MaxDate and CalendarBase.MinDate properties:

XAML

```
<c1:C1Calendar HorizontalAlignment="Left" Margin="10,10,0,0" Name="c1Calendar1"
VerticalAlignment="Top" MaxDate="2099-12-31" MinDate="1700-01-01" />
```

Showing the Previous and Next Month Days of the Month Area Displ

To show the previous and next month days of the month area, enable the C1Calendar.GenerateAdjacentMonthDays property.

Using Visual Studio

To show the previous and next month days of the month area in C1Calendar in Visual Studio:

1. Add a C1Calendar control to your window and select it.
2. In the Properties window, check the checkbox next to the C1Calendar.GenerateAdjacentMonthDays property.

Using XAML

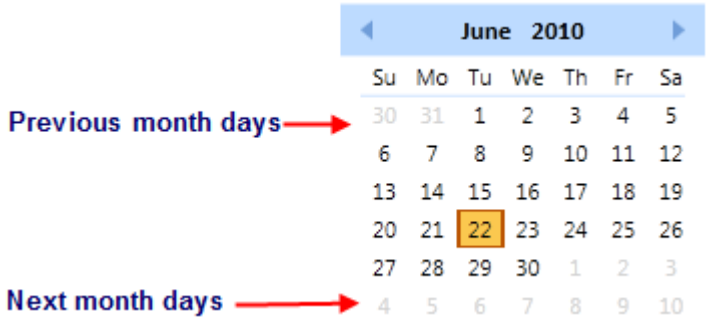
The following XAML sets the C1Calendar.GenerateAdjacentMonthDays property:

XAML

```
<c1:C1Calendar HorizontalAlignment="Left" Margin="10,10,0,0" Name="c1Calendar1"
VerticalAlignment="Top" GenerateAdjacentMonthDays="True" />
```

This topic illustrates the following:

The previous and next month days appear in grey text in the C1Calendar month area.



Specifying the Maximum Number of Days that can be Selected in C1Calendar

To specify the maximum number of days that can be selected in C1Calendar, set the C1Calendar.MaxSelectionCount property to an integer. In this example, we will allow a maximum of 31 days that can be selected in the calendar month display area.

Using Visual Studio

To set the maximum number of selected days to '31' in Visual Studio:

1. Add a C1Calendar control to your window and select it.
2. In the Properties window, enter '31' next to the C1Calendar.MaxSelectionCount property.

Using XAML

The following XAML sets the C1Calendar.MaxSelectionCount property to '31':

XAML

```
<c1:C1Calendar HorizontalAlignment="Left" Margin="10,10,0,0" Name="c1Calendar1"
VerticalAlignment="Top" MaxSelectionCount="31" />
```