ComponentOne

Scheduler for WPF

Corporate Headquarters
ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 · USA

Internet: info@ComponentOne.com
Web site: http://www.componentone.com

Sales
E-mail: sales@componentone.com
Telephone: 1.800.858.2739 or 1.412.681.4343 (Pittsburgh, PA USA Office)

Trademarks

The ComponentOne product name is a trademark and ComponentOne is a registered trademark of ComponentOne LLC. All other trademarks used herein are the properties of their respective owners.

Warranty

ComponentOne warrants that the original CD (or diskettes) are 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 CD (or disk) 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 a defective CD (or disk) by sending it and a check for $25 (to cover postage and handling) to ComponentOne.

Except for the express warranty of the original CD (or disks) 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. We are 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

While you are welcome to make backup copies of the software for your own use and protection, you are not permitted to make copies for the use of anyone else. We put a lot of time and effort into creating this product, and we appreciate your support in seeing that it is used by licensed users only.

This manual was produced using ComponentOne Doc-To-Help™.

Table of Contents

ComponentOne Scheduler for WPF Overview

Create highly sophisticated calendars and schedules in your WPF applications with **ComponentOne Scheduler for WPF**. **Scheduler for WPF** offers full scheduling functionality in one control, C1Scheduler, along with many of the same features included in **ComponentOne Scheduler for ASP.NET AJAX**. Display shared or private appointment calendars in day, week, work week, or monthly views, complete with appointment pop-up reminders, labels, and availability status.

Scheduler for WPF provides the power to partially or completely customize your application's user interface. Choose from a variety of Office 2007 -style themes and use the **Scheduler for WPF** templates to create your own Appointment, Edit, or Reminders dialog boxes.

In addition to C1Scheduler, **Scheduler for WPF** provides one calendar control, C1Calendar, allowing users to select a specific date which can be linked to a schedule.

Smoothly integrate scheduling functionality into your WPF applications to help users stay organized, on time, and ready for upcoming events with **Scheduler for WPF**..

Getting Started

To get started, review the following topics:

- C1Scheduler Key Features (page 15)
- C1Calendar Key Features (page 16)
- C1Scheduler Tasks (page 96)
- C1Calendar Tasks (page 104)

What's New in Scheduler for WPF

This documentation was last revised for 2011 v1 on March 1, 2011.

The following enhancements have been added to **Scheduler for WPF** in the 2011 v1 release of the ComponentOne Studios.

- Default styles and templates have been optimized for better performance.
- Improved Appointment layout.

  CoverElementsPane.CoverElementsMargin property honored for the last cover element only. For example, there is no vertical space between appointments in the Month view and in the All-Day area in all other views, and there is no horizontal space between appointments in Day/Working Week/Week views. If you need horizontal/vertical spaces between appointments, change the **CoverElementsPane.CoverElementsHeight** property value and set appointment margins in the IntervalAppointmentTemplate. **CoverElementsPane.CoverElementsHeight** property default value has been changed to 19.

Tip: A version history containing a list of new features, improvements, fixes, and changes for each product is available in HelpCentral at http://helpcentral.componentone.com/VersionHistory.aspx.

Revision History

The revision history provides recent enhancements to **Scheduler for WPF**.

What's New in 2010 v3

- **ComponentOne Scheduler for WPF** now features ComponentOne's ClearStyle technology, a new and simple approach to providing Silverlight and WPF control styling. For more information about ClearStyle technology, see the **ComponentOne ClearStyle Technology** topic.
- Support for grouping has been added to **Scheduler for WPF**. The C1Scheduler control supports grouping by resources, contacts, categories and by the Owner property value. All default C1Scheduler styles support grouping and contain a UI for navigation between groups. The new **Grouping** sample demonstrates the C1Scheduler grouping functionality. You can also see the Grouping (page 103) task-based help topic for an example.
- Default printing styles, the **PrintDocTemplates** sample, and the **Printing** sample have been updated to honor the new grouping feature.
- A new resource string has been added to the C1.Schedule.MiscStrings.resx file:
  1. MoreAppointments: "Click for more appointments" - the tooltip text for the overlow jumper button in the Month View.
  2. EmptyGroupName: "Calendar" - the default group name which is used for the empty group.

  Note: If you use localized version of the C1.Schedule.MiscStrings.resx file, make sure that you added new strings there. Otherwise, you will receive a runtime MissingManifestResourceException exception.

- The default settings for the **Week** and **Working Week** styles have been optimized for better performance; the **VisualIntervalScale** is 30 minutes.
- The following **C1Scheduler** properties have been marked as obsolete: VisualIntervals, VisualIntervalGroups, and VisualIntervalsView. Use the corresponding SchedulerGroupItem.VisualIntervals, SchedulerGroupItem.VisualIntervalGroups, and SchedulerGroupItem.VisualIntervalsView properties of the SchedulerGroupItem class.

Breaking changes

- The **C1Scheduler** FirstVisibleTime and AutoScrollToFirstAppointment properties have been moved to the C1SchedulerSettings class.
- The AppointmentsCoverPane.KeepTimesAtDrop and AppointmentFilter properties should be set for the AppointmentsCoverPane object directly, not for the VisualIntervalTemplates as in the previous version. This affects custom C1Scheduler control templates.
- The **C1.WPF.Schedule.VTreeHelper** class has been removed. Use **C1.WPF.VTreeHelper** class instead.

Class Member

Member	Description
DaySlot.IsToday Property	Indicates whether the **DaySlot** represents the current day.
Appointment.Owner Property	Gets or sets the **Contact** object which owns current **Appointment** object.
SchedulerGroupItem Class	Holds all the data required for displaying individual UI part for the single resource, category or contact when the **GroupBy** property is set, or default UI otherwise.

SchedulerGroupItemCollection Class	Represents a collection of the **SchedulerGroupItem** objects.
C1Scheduler.GroupBy Property	Gets or sets the String value determining the type of grouping.
C1Scheduler.ShowEmptyGroupItem	Gets or sets the Boolean value determining whether the **C1Scheduler** control should display an empty group item.
C1Scheduler.GroupItems Property	Gets a collection of all available **SchedulerGroupItem** objects for the currently set type of grouping.
C1Scheduler.VisibleGroupItems Property	Gets a collection of currently visible **SchedulerGroupItem** objects. This collection can be used in xaml in the **C1Scheduler** control templates.
C1Scheduler.IsGrouped Property	Gets a Boolean value determining whether grouping has been set for the **C1Scheduler** control. This is a dependency property.
C1Scheduler.SelectedGroupItem Property	Gets the selected **SchedulerGroupItem** object or returns null if the selection is empty. This is a dependency property.
C1Scheduler.GroupPageSize Property	Gets or sets the [!:Integer] value determining the maximum number of the **SchedulerGroupItem** objects displayed in UI at the same time. Increasing this value might affect performance. This is a dependency property.
C1Scheduler.CanNavigateNextGroup Property	Gets a Boolean value determining whether the **C1Scheduler** control can be navigated to the next SchedulerGroupItem object. This is a dependency property.
C1Scheduler.GroupHeaderTemplate Property	Gets or sets a DataTemplate that defines a UI representation of the **SchedulerGroupItem** object header. This is a dependency property.
C1Scheduler.NavigateToPreviousGroup Command Field	Defines the command that navigates **C1Scheduler** UI to the previous **SchedulerGroupItem** object.
C1Scheduler.NavigateToNextGroupCommand Field	Defines the command that navigates **C1Scheduler** UI to the next **SchedulerGroupItem** object.
C1Scheduler.HideGroupCommand Field	Defines the command that hides the specified **SchedulerGroupItem** object.

Installing Scheduler for WPF

The following sections provide helpful information on installing **Scheduler for WPF**.

Scheduler for WPF Setup Files

The installation program will create the directory **C:\Program Files\ComponentOne\Studio for WPF**, which contains the following subdirectories:

bin	Contains copies of all ComponentOne binaries (DLLs, EXEs). For **Scheduler for WPF**, the following .dlls are installed: C1.WPF.Schedule.dll C1.WPF.Schedule.Design.dll – This .dll should always be placed in the same folder as the C1.WPF.Schedule.dll; these files should NOT be placed in the GAC. The C1.WPF.Schedule.Design.dll assembly is required by Blend. C1.WPF.Schedule.Expression.Design.dll

	C1.WPF.Schedule.VisualStudio.Design.4.0.dll
C1WPFNewSchedule\XAML	Contains the full XAML definitions of C1Scheduler and C1Calendar styles and templates which can be used for creating your own custom styles and templates.

The **ComponentOne Studio for WPF Help Setup** program installs integrated Microsoft Help 2.0 and Microsoft Help Viewer help to the C:\Program Files\ComponentOne\Studio for WPF directory in the following folders:

H2Help	Contains Microsoft Help 2.0 integrated documentation for all Studio components.
HelpViewer	Contains Microsoft Help Viewer Visual Studio 2010 integrated documentation for all Studio components.

Samples

Samples for the product are installed in the **ComponentOne Samples** folder by default. The path of the **ComponentOne Samples** directory is slightly different on Windows XP and Windows 7/Vista machines:

Windows XP path: C:\Documents and Settings\<username>\My Documents\ComponentOne Samples

Windows 7/Vista path: C:\Users\<username>\Documents\ComponentOne Samples

The **ComponentOne Samples** folder contains the following subdirectories:

Common	Contains support and data files that are used by many of the demo programs.
Studio for WPF\C1WPFSchedule	Contains samples for **Scheduler for WPF**.

Samples can be accessed from the **ComponentOne Control Explorer**. On your desktop, click the **Start** button and then click **All Programs | ComponentOne | Studio for WPF | Control Explorer**.

System Requirements

System requirements include the following:

Operating Systems:	Microsoft Windows® XP with Service Pack 2 (SP2)
	Windows Vista™
	Windows 7
	Windows Server® 2003
	Windows Server 2008
Environments:	.NET Framework 3.5 or later
	Visual Studio® 2005 extensions for .NET Framework 2.0 November 2006 CTP
Microsoft® Expression® Blend Compatibility:	**Scheduler for WPF** has special design-time support for Blend, which includes workarounds for correct XAML serialization and a customizable user interface.

The C1.WPF.Schedule.Design.dll installed with **Scheduler for WPF** should always be placed in the same folder as the C1.WPF.Schedule.dll; the C1.WPF.Schedule.dll and C1.WPF.Schedule.Design.dll should NOT be placed in the GAC. The C1.WPF.Schedule.Design.dll assembly is required by Blend.

Installing Demonstration Versions

If you wish to try **Scheduler for WPF** and do not have a serial number, follow the steps through the installation wizard and use the default serial number.

The only difference between unregistered (demonstration) and registered (purchased) versions of our products is that registered versions will stamp every application you compile so a ComponentOne banner will not appear when your users run the applications.

Uninstalling Scheduler for WPF

To uninstall **ComponentOne Scheduler for WPF**:

1. Open the **Control Panel** and select **Add or Remove Programs** (**Programs and Features** in Windows 7/Vista).
2. Select **ComponentOne Studio for WPF** and click the **Remove** button.
3. Click **Yes** to remove the program.

To uninstall **ComponentOne Scheduler for WPF** integrated help:

1. Open the Control Panel and select **Add or Remove Programs** (**Programs and Features** in Windows 7/Vista).
2. Select **ComponentOne Studio for WPF Help** and click the **Remove** button.
3. Click **Yes** to remove the integrated help.

End-User License Agreement

All of the ComponentOne licensing information, including the ComponentOne end-user license agreements, frequently asked licensing questions, and the ComponentOne licensing model, is available online at http://www.componentone.com/SuperPages/Licensing/.

Licensing FAQs

This section describes the main technical aspects of licensing. It may help the user to understand and resolve licensing problems he may experience when using ComponentOne .NET and ASP.NET products.

What is Licensing?

Licensing is a mechanism used to protect intellectual property by ensuring that users are authorized to use software products.

Licensing is not only used to prevent illegal distribution of software products. Many software vendors, including ComponentOne, use licensing to allow potential users to test products before they decide to purchase them.

Without licensing, this type of distribution would not be practical for the vendor or convenient for the user. Vendors would either have to distribute evaluation software with limited functionality, or shift the burden of managing software licenses to customers, who could easily forget that the software being used is an evaluation version and has not been purchased.

How does Licensing Work?

ComponentOne uses a licensing model based on the standard set by Microsoft, which works with all types of components.

> **Note:** The **Compact Framework** components use a slightly different mechanism for run-time licensing than the other ComponentOne components due to platform differences.

When a user decides to purchase a product, he receives an installation program and a Serial Number. During the installation process, the user is prompted for the serial number that is saved on the system. (Users can also enter the serial number by clicking the **License** button on the **About Box** of any ComponentOne product, if available, or by rerunning the installation and entering the serial number in the licensing dialog box.)

When a licensed component is added to a form or Web page, Visual Studio obtains version and licensing information from the newly created component. When queried by Visual Studio, the component looks for licensing information stored in the system and generates a run-time license and version information, which Visual Studio saves in the following two files:

- An assembly resource file which contains the actual run-time license
- A "licenses.licx" file that contains the licensed component strong name and version information

These files are automatically added to the project.

In WinForms and ASP.NET 1.x applications, the run-time license is stored as an embedded resource in the assembly hosting the component or control by Visual Studio. In ASP.NET 2.x applications, the run-time license may also be stored as an embedded resource in the App_Licenses.dll assembly, which is used to store all run-time licenses for all components directly hosted by WebForms in the application. Thus, the App_licenses.dll must always be deployed with the application.

The licenses.licx file is a simple text file that contains strong names and version information for each of the licensed components used in the application. Whenever Visual Studio is called upon to rebuild the application resources, this file is read and used as a list of components to query for run-time licenses to be embedded in the appropriate assembly resource. Note that editing or adding an appropriate line to this file can force Visual Studio to add run-time licenses of other controls as well.

Note that the licenses.licx file is usually not shown in the Solution Explorer; it appears if you press the **Show All Files** button in the Solution Explorer's toolbox, or from Visual Studio's main menu, select **Show All Files** on the **Project** menu.

Later, when the component is created at run time, it obtains the run-time license from the appropriate assembly resource that was created at design time and can decide whether to simply accept the run-time license, to throw an exception and fail altogether, or to display some information reminding the user that the software has not been licensed.

All ComponentOne products are designed to display licensing information if the product is not licensed. None will throw licensing exceptions and prevent applications from running.

Common Scenarios

The following topics describe some of the licensing scenarios you may encounter.

Creating components at design time

This is the most common scenario and also the simplest: the user adds one or more controls to the form, the licensing information is stored in the licenses.licx file, and the component works.

Note that the mechanism is exactly the same for Windows Forms and Web Forms (ASP.NET) projects.

Creating components at run time

This is also a fairly common scenario. You do not need an instance of the component on the form, but would like to create one or more instances at run time.

In this case, the project will not contain a licenses.licx file (or the file will not contain an appropriate run-time license for the component) and therefore licensing will fail.

To fix this problem, add an instance of the component to a form in the project. This will create the licenses.licx file and things will then work as expected. (The component can be removed from the form after the licenses.licx file has been created).

Adding an instance of the component to a form, then removing that component, is just a simple way of adding a line with the component strong name to the licenses.licx file. If desired, you can do this manually using notepad or Visual Studio itself by opening the file and adding the text. When Visual Studio recreates the application resources, the component will be queried and its run-time license added to the appropriate assembly resource.

Inheriting from licensed components

If a component that inherits from a licensed component is created, the licensing information to be stored in the form is still needed. This can be done in two ways:

- Add a LicenseProvider attribute to the component.

  This will mark the derived component class as licensed. When the component is added to a form, Visual Studio will create and manage the licenses.licx file, and the base class will handle the licensing process as usual. No additional work is needed. For example:

```
[LicenseProvider(typeof(LicenseProvider))]
class MyGrid: C1.Win.C1FlexGrid.C1FlexGrid
{
  // ...
}
```

- Add an instance of the base component to the form.

  This will embed the licensing information into the licenses.licx file as in the previous scenario, and the base component will find it and use it. As before, the extra instance can be deleted after the licenses.licx file has been created.

Please note, that C1 licensing will not accept a run-time license for a derived control if the run-time license is embedded in the same assembly as the derived class definition, and the assembly is a DLL. This restriction is necessary to prevent a derived control class assembly from being used in other applications without a design time license. If you create such an assembly, you will need to take one of the actions previously described create a component at run time.

Using licensed components in console applications

When building console applications, there are no forms to add components to, and therefore Visual Studio won't create a licenses.licx file.

In these cases, create a temporary Windows Forms application and add all the desired licensed components to a form. Then close the Windows Forms application and copy the licenses.licx file into the console application project.

Make sure the licenses.licx file is configured as an embedded resource. To do this, right-click the licenses.licx file in the Solution Explorer window and select **Properties**. In the property window, set the **Build Action** property to **Embedded Resource**.

Using licensed components in Visual C++ applications

There is an issue in VC++ 2003 where the licenses.licx is ignored during the build process; therefore, the licensing information is not included in VC++ applications.

To fix this problem, extra steps must be taken to compile the licensing resources and link them to the project. Note the following:

1. Build the C++ project as usual. This should create an exe file and also a licenses.licx file with licensing information in it.
2. Copy the licenses.licx file from the app directory to the target folder (Debug or Release).
3. Copy the C1Lc.exe utility and the licensed dlls to the target folder. (Don't use the standard lc.exe, it has bugs.)
4. Use C1Lc.exe to compile the licenses.licx file. The command line should look like this:
   ```
   c1lc /target:MyApp.exe /complist:licenses.licx /i:C1.Win.C1FlexGrid.dll
   ```
5. Link the licenses into the project. To do this, go back to Visual Studio, right-click the project, select properties, and go to the Linker/Command Line option. Enter the following:
   ```
   /ASSEMBLYRESOURCE:Debug\MyApp.exe.licenses
   ```
6. Rebuild the executable to include the licensing information in the application.

Using licensed components with automated testing products

Automated testing products that load assemblies dynamically may cause them to display license dialog boxes. This is the expected behavior since the test application typically does not contain the necessary licensing information, and there is no easy way to add it.

This can be avoided by adding the string "C1CheckForDesignLicenseAtRuntime" to the AssemblyConfiguration attribute of the assembly that contains or derives from ComponentOne controls. This attribute value directs the ComponentOne controls to use design time licenses at run time.

For example:

```
#if AUTOMATED_TESTING
      [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime")]
#endif
      public class MyDerivedControl : C1LicensedControl
      {
            // ...
      }
```

Note that the AssemblyConfiguration string may contain additional text before or after the given string, so the AssemblyConfiguration attribute can be used for other purposes as well. For example:

```
      [AssemblyConfiguration("C1CheckForDesignLicenseAtRuntime,BetaVersion")]
```

THIS METHOD SHOULD ONLY BE USED UNDER THE SCENARIO DESCRIBED. It requires a design time license to be installed on the testing machine. Distributing or installing the license on other computers is a violation of the EULA.

Troubleshooting

We try very hard to make the licensing mechanism as unobtrusive as possible, but problems may occur for a number of reasons.

Below is a description of the most common problems and their solutions.

I have a licensed version of a ComponentOne product but I still get the splash screen when I run my project.

If this happens, there may be a problem with the licenses.licx file in the project. It either doesn't exist, contains wrong information, or is not configured correctly.

First, try a full rebuild (**Rebuild All** from the Visual Studio **Build** menu). This will usually rebuild the correct licensing resources.

If that fails follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and open it. If prompted, continue to open the file.
4. Change the version number of each component to the appropriate value. If the component does not appear in the file, obtain the appropriate data from another licenses.licx file or follow the alternate procedure following.
5. Save the file, then close the licenses.licx tab.
6. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

Alternatively, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Click the **Show All Files** button on the top of the window.
3. Find the licenses.licx file and delete it.
4. Close the project and reopen it.
5. Open the main form and add an instance of each licensed control.
6. Check the Solution Explorer window, there should be a licenses.licx file there.
7. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

For ASP.NET 2.x applications, follow these steps:

1. Open the project and go to the Solution Explorer window.
2. Find the licenses.licx file and right-click it.
3. Select the **Rebuild Licenses** option (this will rebuild the App_Licenses.licx file).
4. Rebuild the project using the **Rebuild All** option (not just **Rebuild**).

I have a licensed version of a ComponentOne product on my Web server but the components still behave as unlicensed.

There is no need to install any licenses on machines used as servers and not used for development.

The components must be licensed on the development machine, therefore the licensing information will be saved into the executable (.exe or .dll) when the project is built. After that, the application can be deployed on any machine, including Web servers.

For ASP.NET 2.x applications, be sure that the App_Licenses.dll assembly created during development of the application is deployed to the bin application bin directory on the Web server.

If your ASP.NET application uses WinForms user controls with constituent licensed controls, the runtime license is embedded in the WinForms user control assembly. In this case, you must be sure to rebuild and update the user control whenever the licensed embedded controls are updated.

I downloaded a new build of a component that I have purchased, and now I'm getting the splash screen when I build my projects.

Make sure that the serial number is still valid. If you licensed the component over a year ago, your subscription may have expired. In this case, you have two options:

Option 1 – Renew your subscription to get a new serial number.

If you choose this option, you will receive a new serial number that you can use to license the new components (from the installation utility or directly from the **About Box**).

The new subscription will entitle you to a full year of upgrades and to download the latest maintenance builds directly from http://prerelease.componentone.com/.

Option 2 – Continue to use the components you have.

Subscriptions expire, products do not. You can continue to use the components you received or downloaded while your subscription was valid.

Technical Support

ComponentOne offers various support options. For a complete list and a description of each, visit the ComponentOne Web site at http://www.componentone.com/SuperProducts/SupportServices/.

Some methods for obtaining technical support include:

- **Online Resources**

  ComponentOne provides customers with a comprehensive set of technical resources in the form of FAQs, samples and videos, Version Release History, searchable Knowledge base, searchable Online Help and more. We recommend this as the first place to look for answers to your technical questions.

- **Online Support via our Incident Submission Form**

  This online support service provides you with direct access to our Technical Support staff via an online incident submission form. When you submit an incident, you'll immediately receive a response via e-mail confirming that you've successfully created an incident. This e-mail will provide you with an Issue Reference ID and will provide you with a set of possible answers to your question from our Knowledgebase. You will receive a response from one of the ComponentOne staff members via e-mail in 2 business days or less.

- **Peer-to-Peer Product Forums**

  ComponentOne peer-to-peer product forums are available to exchange information, tips, and techniques regarding ComponentOne products. ComponentOne sponsors these areas as a forum for users to share information. While ComponentOne does not provide direct support in the forums, we periodically monitor them to ensure accuracy of information and provide comments when appropriate. Please note that a ComponentOne User Account is required to participate in the ComponentOne Product Forums.

- **Installation Issues**

  Registered users can obtain help with problems installing ComponentOne products. Contact technical support by using the online incident submission form or by phone (412.681.4738). Please note that this does not include issues related to distributing a product to end-users in an application.

- **Documentation**

  Microsoft integrated ComponentOne documentation can be installed with each of our products, and documentation is also available online. If you have suggestions on how we can improve our documentation, please email the Documentation team. Please note that e-mail sent to the Documentation team is for documentation feedback only. Technical Support and Sales issues should be sent directly to their respective departments.

Note: You must create a ComponentOne Account and register your product with a valid serial number to obtain support using some of the above methods.

Redistributable Files

ComponentOne **Scheduler for WPF** is developed and published by ComponentOne LLC. You may use it to develop applications in conjunction with Microsoft Visual Studio or any other programming environment that enables the user to use and integrate the control(s). You may also distribute, free of royalties, the following

Redistributable Files with any such application you develop to the extent that they are used separately on a single CPU on the client/workstation side of the network:

- C1.WPF.Schedule.dll
- C1.C1Report.2.dll

Site licenses are available for groups of multiple developers. Please contact Sales@ComponentOne.com for details.

About This Documentation

You can create your schedule applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, we will use the **Design** workspace of Blend for most examples.

Acknowledgements

ComponentOne

If you have any suggestions or ideas for new features or controls, please call us or write:

Corporate Headquarters

ComponentOne LLC
201 South Highland Avenue
3rd Floor
Pittsburgh, PA 15206 • USA
412.681.4343
412.681.4384 (Fax)
http://www.componentone.com/

ComponentOne Doc-To-Help

This documentation was produced using ComponentOne Doc-To-Help® Enterprise.

XAML and XAML Namespaces

XAML is a declarative XML-based language that is used as a user interface markup language in Windows Presentation Foundation and the .NET Framework 3.0. With XAML you can create a graphically rich customized user interface, perform databinding, and much more. For more information on XAML and the .NET Framework 3.0, please see http://www.microsoft.com.

XAML Namespaces

Namespaces organize the objects defined in an assembly. Assemblies can contain multiple namespaces, which can in turn contain other namespaces. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

When you create a Microsoft Expression Blend project, a XAML file is created for you, and some initial namespaces are specified:

Namespace	Description
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"	This is the default Windows Presentation Foundation namespace.
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"	This is a XAML namespace that is mapped to the **x:** prefix. The **x:** prefix provides a quick, easy way to reference the namespace, which defines many commonly-used features necessary for WPF applications.

When you add a C1Scheduler control to the window, **Blend** automatically creates a namespace that looks like the following:

```
xmlns:c1sched="http://schemas.componentone.com/wpf/C1Schedule"
```

You can also choose to create your own custom name for the namespace. For example:

```
xmlns:MyC1Schedule="http://schemas.componentone.com/wpf/C1Schedule"
```

You can now use your custom namespace when assigning properties, methods and events. For example, to set the **C1Scheduler.WeekStyle** property, use the following XAML:

```
<MyC1Schedule:C1Scheduler Style="{x:Static
MyC1Schedule:C1Scheduler.WeekStyle}"/>
```

Note: We will use the **c1sched** namespace throughout the documentation for consistency, but you may use any custom namespace desired.

Creating a Microsoft Expression Blend Project

To create a new Blend project, start Expression Blend and complete the following steps:

1. From the **File** menu, select **New Project** or click **New Project** in the Blend startup window. The **Create New Project** dialog box opens.
2. Make sure **WPF Application (.exe)** is selected and enter a name for the project in the **Name** text box.
3. Browse to specify a location for the project.
4. Select a language from the **Language** drop-down box and click **OK**. A new Blend project with a XAML window is created.

Creating a .NET Project in Visual Studio

To create a new .NET project in Visual Studio 2010, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2010, select **New Project**. The **New Project** dialog box opens.
2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.
3. Under **Project types**, select either **Visual Basic** or **Visual C#**.
4. Choose **WPF Application** from the list of **Templates** in the right pane.
5. Enter a name for your application in the **Name** field and click **OK**.

   A new Microsoft Visual Studio .NET WPF project is created with a XAML file that will be used to define your user interface and commands in the application.

   Note: You can create your grid applications using Microsoft Expression Blend or Visual Studio, but Blend is currently the only design-time environment that allows users to design XAML documents visually. In this documentation, Blend will be used for most examples.

Creating an XAML Browser Application (XBAP) in Visual Studio

To create a new XAML Browser Application (XBAP) in Visual Studio 2008, complete the following steps:

1. From the **File** menu in Microsoft Visual Studio 2008, select **New Project**. The **New Project** dialog box opens.

2. Choose the appropriate .NET Framework from the Framework drop-down box in the top-right of the dialog box.
3. Under Project types, select either **Visual Basic** or **Visual C#** and then select **Windows**.
4. Choose **WPF Browser Application** from the list of **Templates** in the right pane.

   Note: If using Visual Studio 2005, you may need to select **XAML Browser Application (WPF)** after selecting **NET Framework 3.0** under **Visual Basic** or **Visual C#** in the left-side menu.

5. Enter a name for your application in the **Name** field and click **OK**.

   A new Microsoft Visual Studio .NET WPF Browser Application project is created with a XAML file that will be used to define your user interface and commands in the application.

Adding the Scheduler for WPF Controls to a Blend Project

In order to use C1Scheduler or C1Calendar another **Scheduler for WPF** control in the Design workspace of Blend, you must first add a reference to the C1.WPF.C1Schedule assembly and then add the component from Blend's **Asset Library**.

To add a reference to the assembly:

1. Select **Project | Add Reference**.
2. Browse to find the C1.WPF.Schedule.dll installed with **Scheduler for WPF**.

   Note: Both the C1.WPF.Schedule.dll and C1.WPF.Schedule.Design.dll files are installed to C:\Program Files\ComponentOne\Studio for WPF\bin by default. The C1.WPF.Schedule.Design.dll installed with **Scheduler for WPF** should always be placed in the same folder as the C1.WPF.Schedule.dll. It should NOT be placed in the GAC. This assembly is required by Blend.

3. Select **C1.WPF.Schedule.dll** and click **Open**. A reference is added to your project.

To add a component from the Asset Library:

1. Once you have added a reference to the C1.WPF.Schedule assembly, click the **Asset Library** button in the Blend Toolbox. The **Asset Library** appears.
2. Click the **Categories** tab. All of the **Scheduler for WPF** components are listed here.
3. Select **C1Scheduler** or **C1Calendar**. The **C1Scheduler** or **C1Calendar** component will appear in the Toolbox above the **Asset Library** button.
4. Double-click the **C1Scheduler** or **C1Calendar** component in the Toolbox to add it to the Window1.xaml.

Adding the Scheduler for WPF Components to a Visual Studio Project

When you install **Scheduler for WPF**, the C1Scheduler and C1Calendar controls should be added to your Visual Studio Toolbox. You can also manually add ComponentOne controls to the Toolbox.

ComponentOne Scheduler for WPF provides the following controls:

- C1Scheduler
- C1Calendar

To use **Scheduler for WPF**, add this control to the window or add a reference to the **C1.WPF.Schedule** assembly to your project.

Manually Adding Scheduler for WPF to the Toolbox

When you install **Scheduler for WPF**, the following **Scheduler for WPF** component will appear in the Visual Studio Toolbox customization dialog box:

- C1Scheduler
- C1Calendar

To manually add the C1Scheduler control to the Visual Studio Toolbox, complete the following steps:

1. Open the Visual Studio IDE (Microsoft Development Environment). Make sure the Toolbox is visible (select **Toolbox** in the **View** menu, if necessary) and right-click the Toolbox to open its context menu.
2. To make the **C1Scheduler** component appear on its own tab in the Toolbox, select **Add Tab** from the context menu and type in the tab name, **C1Scheduler**, for example.
3. Right-click the tab where the component is to appear and select **Choose Items** from the context menu. The **Choose Toolbox Items** dialog box opens.
4. In the dialog box, select the **WPF Components** tab. Sort the list by Namespace (click the *Namespace* column header) and select the check boxes for all components belonging to the C1.WPF.C1Schedule namespace. Note that there may be more than one component for each namespace.

Adding C1Scheduler or C1Calendar to the Window

To add C1Scheduler or C1Calendar to a window or page, complete the following steps:

1. Add the C1Scheduler or C1Calendar control to the Visual Studio Toolbox.
2. Double-click C1Scheduler or C1Calendar drag the control onto the window.

Adding a Reference to the Assembly

To add a reference to the **C1.WPF.Schedule** assembly, complete the following steps:

1. Select the **Add Reference** option from the **Project** menu of your project.
2. Select the **ComponentOne WPF Scheduler** assembly from the list on the **.NET** tab or on the **Browse** tab, browse to find the C1.WPF.Schedule.dll assembly and click **OK**.
3. Double-click the window caption area to open the code window. At the top of the file, add the following **Imports** statements (**using** in C#):
   ```
   Imports C1.WPF.Schedule
   ```

   This makes the objects defined in the **C1.WPF.Schedule** assembly visible to the project.

C1Scheduler Key Features

ComponentOne Scheduler for WPF is a set of controls that allows you to integrate scheduling functionality into your Windows Presentation Foundation (WPF) applications. It follows the "lookless control" approach introduced by WPF, meaning just as you design your own user interface (UI), or look and feel, for WPF applications, you can provide your own UI for data managed by **Scheduler for WPF**. Extensible Application Markup Language (XAML; pronounced "Zammel"), an XML-based declarative language, offers a simple approach to designing your UI without having to write code, although anything written in XAML can also be defined in standard .NET languages such as Visual Basic .NET and C#. Most of the views and dialog boxes provided by **Scheduler for WPF**, with very rare exceptions, are implemented fully in XAML. Control behavior can be implemented using code.

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

- **Standard Data Binding or Built-in Data Source**

  Scheduler for WPF gives you the option of using standard ADO.NET data binding 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, labels, resources, and the status of appointments.

- **Import and Export Multiple Formats**

  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** Export and Import methods or using C1Scheduler routed commands, ExportCommand and ImportCommand.

- **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.

- **Create Customized Dialog Boxes**

  Scheduler for WPF 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 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 (page 34) for more information on **Scheduler for WPF** templates.

- **Built-in Data Views**

  Scheduler for WPF includes four built-in data views, allowing you to offer a variety of ways for users to view their schedules. See C1Scheduler Data Views (page 30) for more information on changing the view.

- **Office 2007 -style Themes**

  Scheduler for WPF includes fourteen 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 Themes (page 35) or C1Calendar Themes (page 69).

C1Calendar Key Features

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

- **Multi-Month Views**

  C1Calendar supports the display of multiple months. The number of months viewable is determined by the MonthCount property and the size of the control's container. Display multiple months horizontally, vertically, or both ways by simply adjusting the size of the control. For more information see Multi-Month Calendar Display (page 74).

- **Interactive Navigation**

  Users can interact with C1Calendar to navigate months by clicking the arrow buttons on the month header. Users can also conveniently navigate to a different month or year by clicking the drop-downs associated with each.

- **Multiple Date Selection**

  Allow users to select any number of days. Control exactly how many days can be selected by setting the MaxSelectionCount property. For more information on multiple date selection, see

- **Easy and Flexible Styling Model**

  C1Calendar exposes brushes for each visible part of the control allowing you to fully customize the style without having to modify the control's template. The styleable parts include:

  - Background/Border
  - Month Header
  - Navigation Buttons
  - Days of Week
  - Selected Day
  - Today's Date
  - Weekend Days
  - Adjacent Month Days

  For more information on C1Calendar's available brushes

- **Synchronize with C1Scheduler**

  Easily integrate C1Calendar with C1Scheduler for a fully navigational scheduling solution. C1Calendar and C1Scheduler share the same code base for all calendar related settings.

- **Globalization Support**

  C1Calendar uses calendar settings defined for the current thread culture. This includes localized month names, days of week names and abbreviations. You can also customize calendar settings such as week start and the set of working days through the CalendarHelper class. For more information on C1Calendar's settings, see

Scheduler for 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 C:\Program Files\ComponentOne\Studio for WPF\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:
    1. Select **File | New Project**. The **New Project** dialog box opens.
    2. 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#**.
    3. Choose WPF Application from the list of *Templates* in the right pane.
    4. 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 WPF** is installed, this file is installed to **C:\Program Files\ComponentOne\Studio for WPF\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:
    ```
    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 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.

```
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.

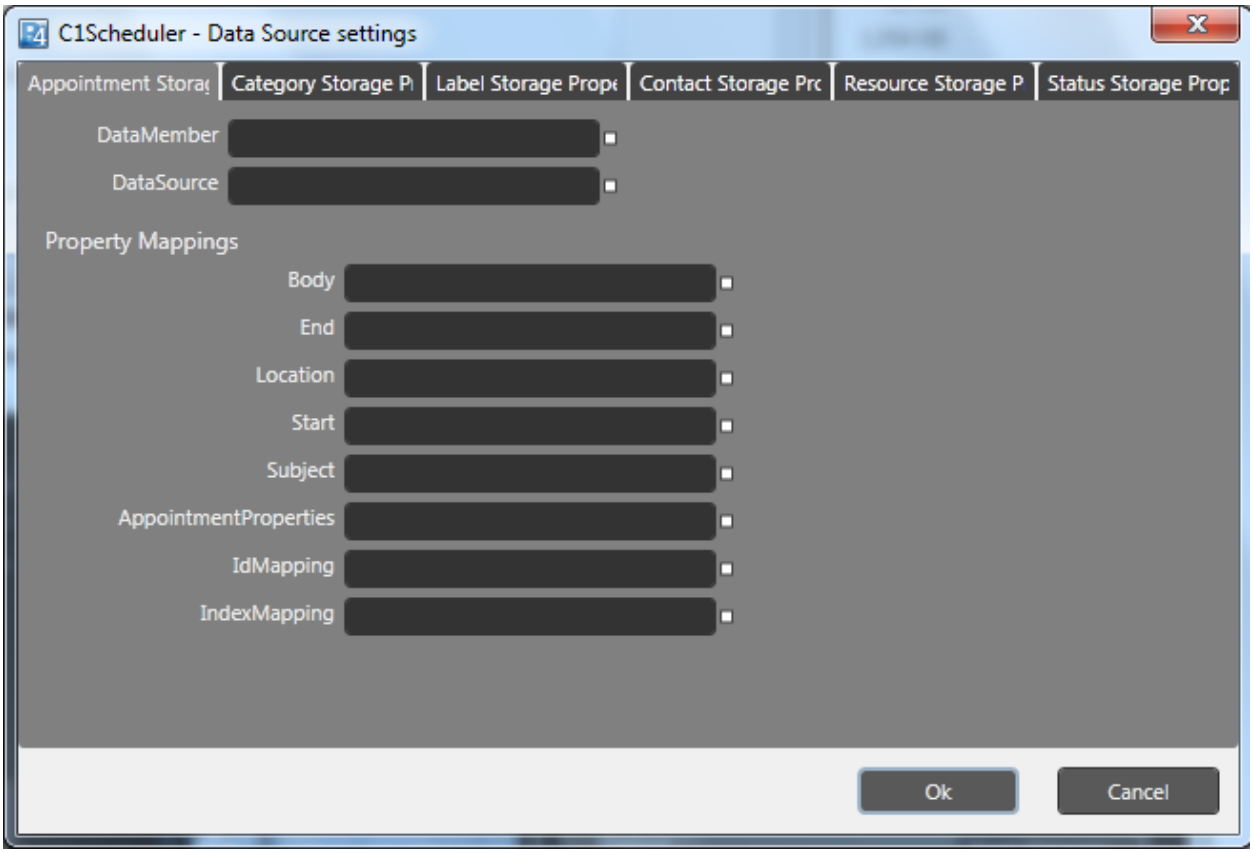
To add the new dataset as a resource to your Blend project:

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.

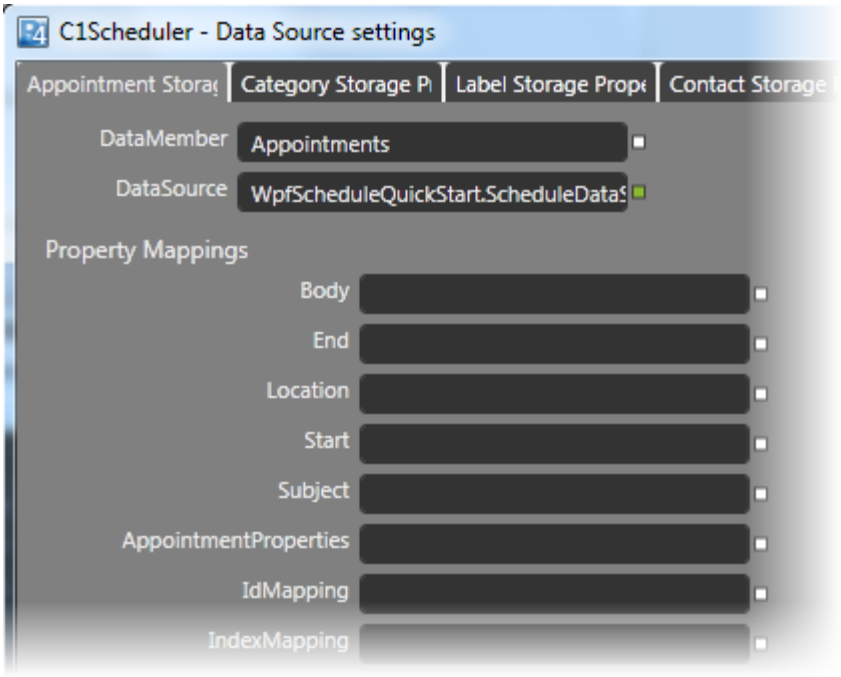
3. Add a C1Scheduler control to the page. Since you have already added a reference to the C1.WPF.C1Schedule assembly when you created the project in Visual Studio 2008, all you need to do is add C1Scheduler from the Asset Library:
   a. Click the **Asset Library** button in the Blend toolbox.
   b. In 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:
   ```
   <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:
   ```
   <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.

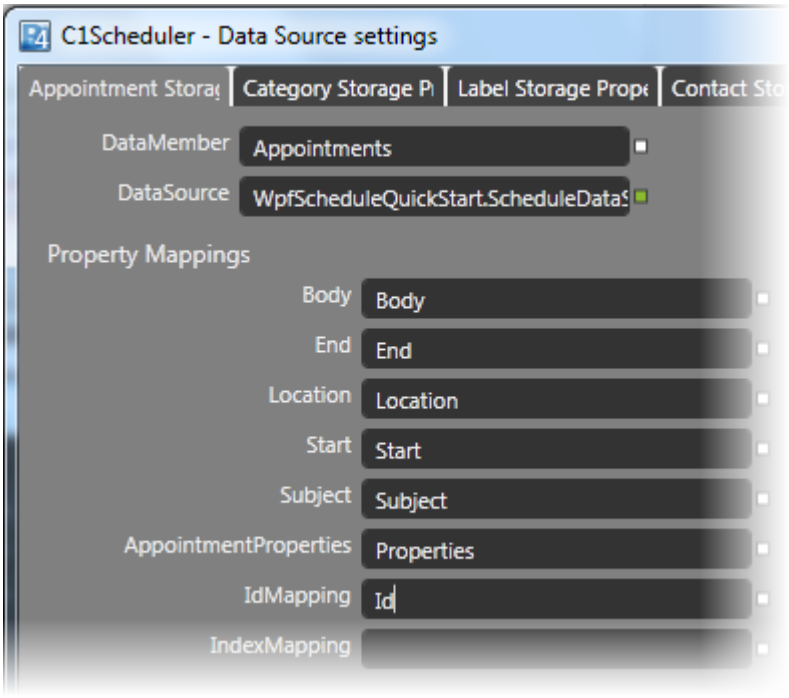


4. Click the **Advanced Property Options** button next to the **DataSource** property, select **Local Resource** and check **dataSet**. This property is set to the **ScheduleDataSet**.
5. Enter **Appointments** next to the **DataMember** property. C1Scheduler will map to the **Appointments** table and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

```
<c1sched:C1Scheduler HorizontalAlignment="Left" Height="400"
VerticalAlignment="Top" Width="480">
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataMember"
Value="Appointments"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataSource"
Value="{DynamicResource dataSet}"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Body.MappingName"
Value="Body"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.End.MappingName"
Value="End"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Location.MappingName"
Value="Location"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Start.MappingName"
Value="Start"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Subject.MappingName"
Value="Subject"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.AppointmentProperties
.MappingName" Value="Properties"/>
            <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.IdMapping.MappingName
" Value="Id"/>
      </c1sched: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 Style property and select **Reset**.
3. Click the drop-down arrow again and select **Week View**.

To select a Theme:

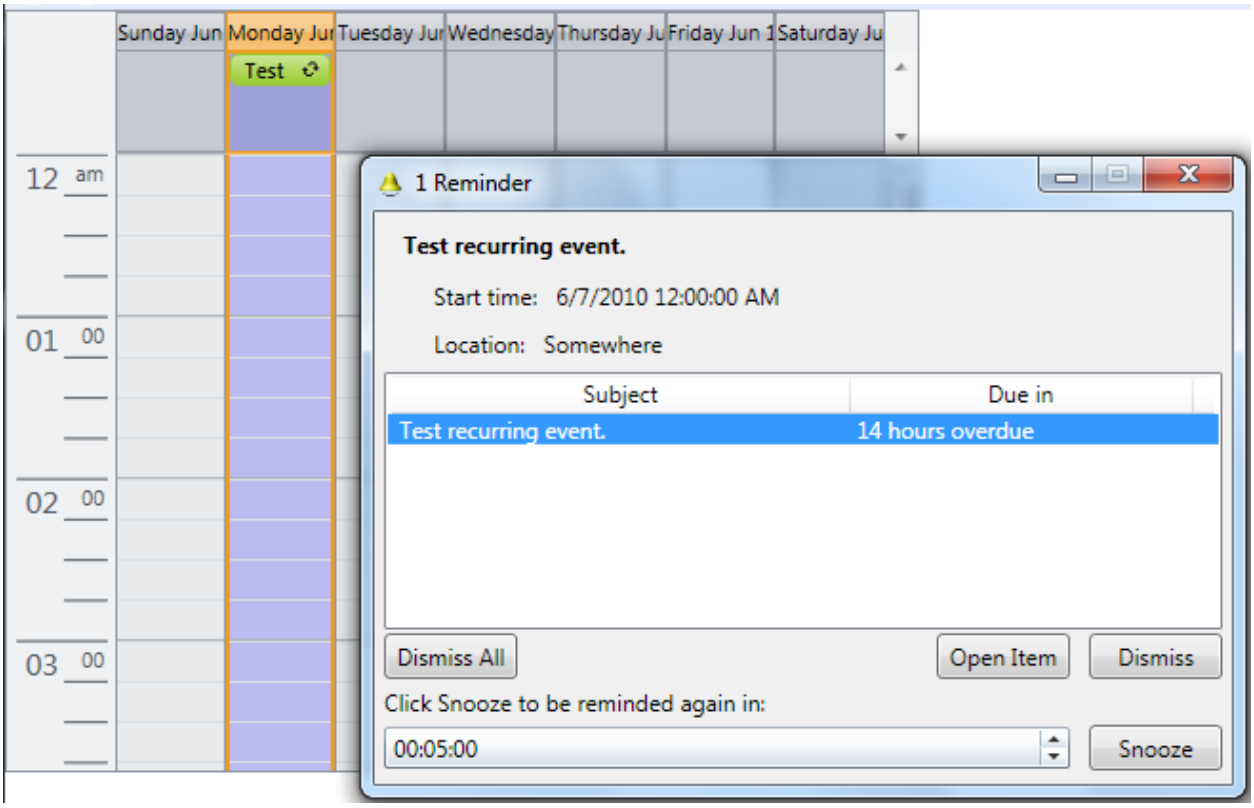
1. Expand the **View** node, click the drop-down arrow next to Theme, and select **Office 2007 Black**.

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.

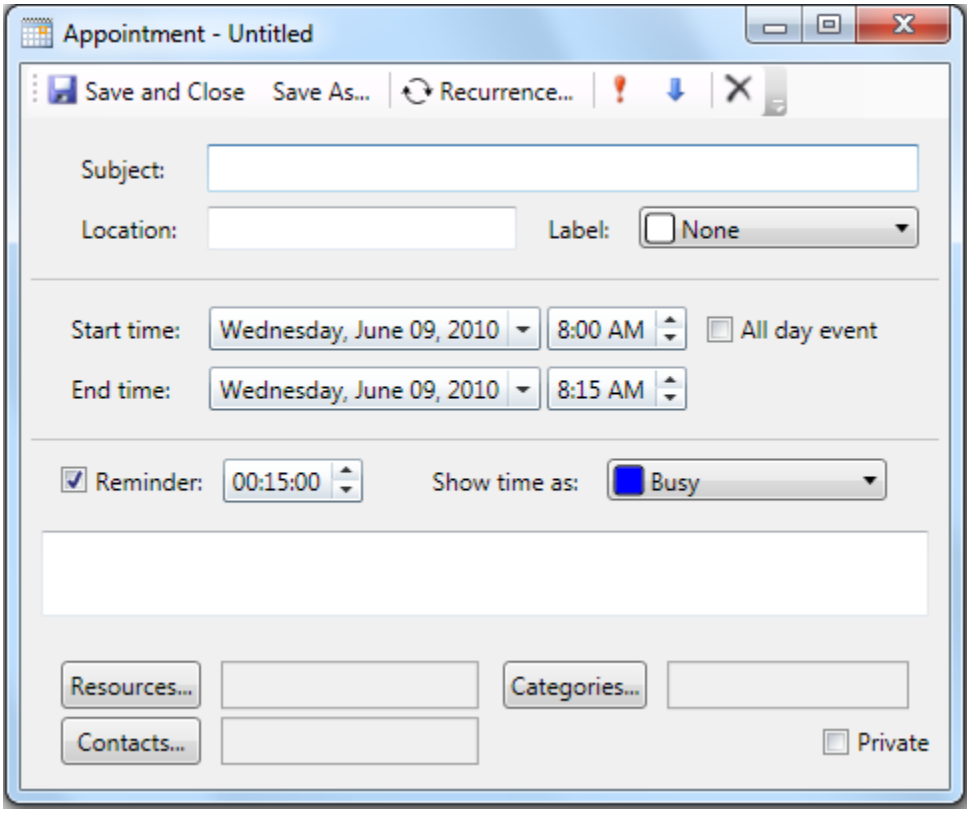
To run the schedule application and observe Scheduler for WPF's run-time behavior:

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

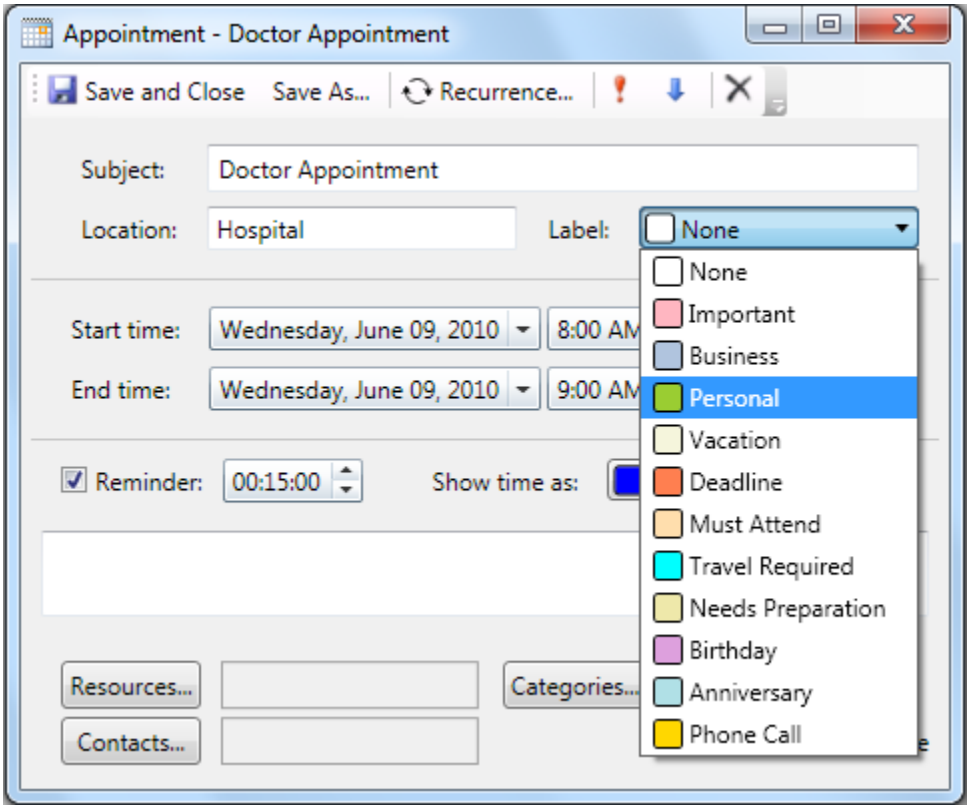


2. Set up an appointment by double-clicking the time for the appointment under the desired day and time. In this example, create an appointment for 8AM on Wednesday, June 9th. The **Appointment** dialog box opens.

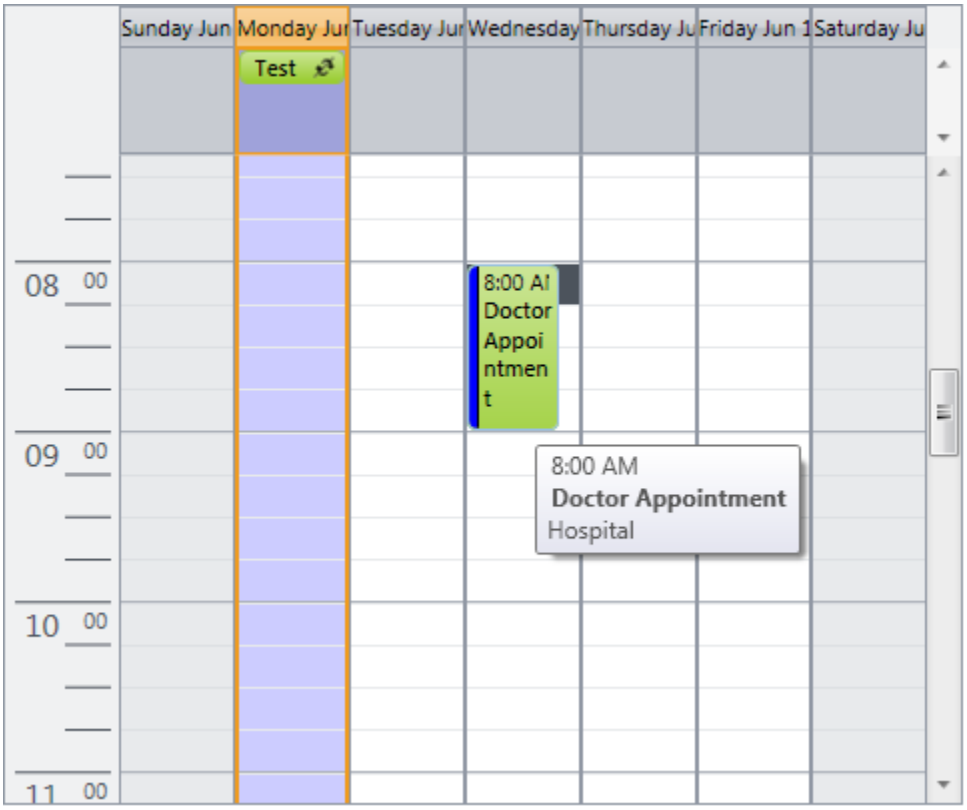
4.



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 **Scheduler for WPF** quick start and created a **Scheduler for WPF** application, bound the schedule to a data source, and created a new appointment.

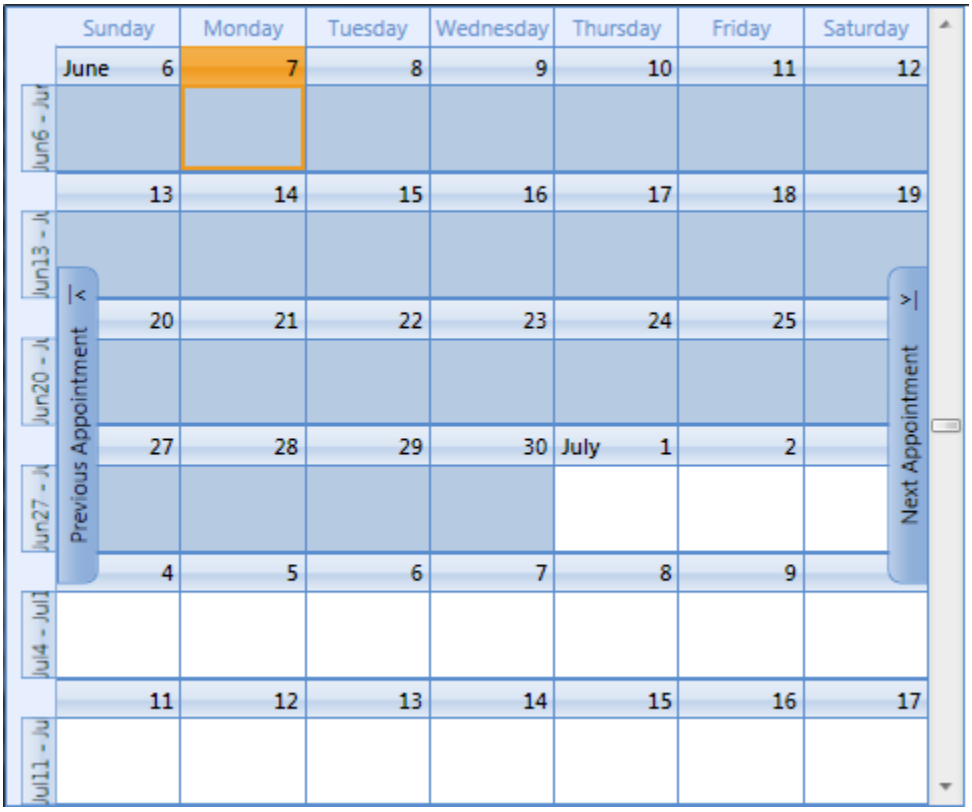
Scheduler for WPF Controls and Components

ComponentOne Scheduler for 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 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.
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 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 DataSource and 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.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.

Scheduler for WPF Design-Time Support

The following sections describe how to use **C1Scheduler**'s design-time environment in Visual Studio to configure the C1Scheduler control.

C1Scheduler Context Menu

C1Scheduler has additional commands available on the context menu that Visual Studio provides for all controls.

To open the C1Scheduler context menu, right-click anywhere on the C1Scheduler control. Options include:

- **About ComponentOne WPF Scheduler**

  Clicking **About ComponentOne WPF Scheduler** displays the **About Scheduler for WPF** dialog box, which is helpful in finding the version number of the product, helpful resources, and other information.

- **View**

Clicking **View** allows you to set the data view to one of the following options: One Day, Work Week, Week, or Month.

- **Theme**

  Clicking **Theme** allows you to choose from seven predefined themes for your schedule. Themes include: Office 2007 Blue (Default), Office 2007 Black, Office 2007 Silver, Media Player, Dusk Blue, Dusk Green, and Vista.

- **Reset Appearance**

  Clicking **Reset Appearance** sets the theme and view back to their default values, Office 2007 Blue and Month, respectively.

C1Calendar Context Menu

The C1Calendar control provides additional commands on the context menu that Visual Studio provides for all controls.

To open the C1Calendar context menu, right-click anywhere on the C1Calendar control. Options include:

- **About ComponentOne Calendar for WPF**

  Clicking **About ComponentOne Calendar for WPF** displays the **About Scheduler for WPF** dialog box, which is helpful in finding the version number of the product, helpful resources, and other information.

- **Theme**

  Clicking **Theme** allows you to choose from seven predefined themes for your schedule. Themes include: Default (Office 2007 Blue), Dusk Blue, Dusk Green, Media Player, Office 2007 Black, Office 2007 Blue, Office 2007 Silver, and Vista.

- **Reset Appearance**

  Clicking **Reset Appearance** sets the theme back to its default value of Office 2007 Blue.

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:

C1Scheduler 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.

C1Scheduler Data Views

Scheduler for WPF provides four views to display data in the 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.

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 the static fields of the C1SchedulerResources class, including: *OneDayStyle*, *WorkingWeekStyle*, *WeekStyle*, and *MonthStyle*. 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 ChangeStyle method using the OneDayStyle field value:

- Visual Basic
```
Scheduler1.ChangeStyle(C1SchedulerResources.OneDayStyle)
```
- C#
```
Scheduler1.ChangeStyle(C1SchedulerResources.OneDayStyle);
```
- XAML
```
<c1sched:C1Scheduler Style="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=OneDayStyle}}"/>
```

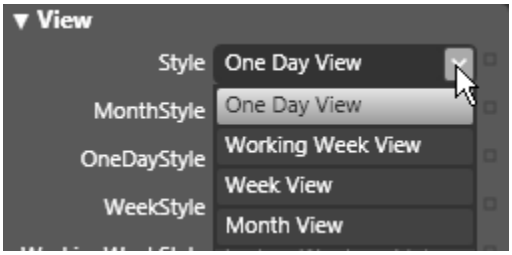
The data view can be set at design time in Visual Studio using the context menu:

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 BeforeViewChange event or by setting the C1Scheduler ViewStyleSelector property to the custom **StyleSelector** object.

By default, these styles are set to styles defined in the default C1Scheduler theme ResourceDictionary. When the Theme property is changed, the control looks for a predefined set of ComponentResourceKeys. If a ComponentResourceKey is found in the new theme resource dictionary, then the corresponding style gets updated with the new one.

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

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

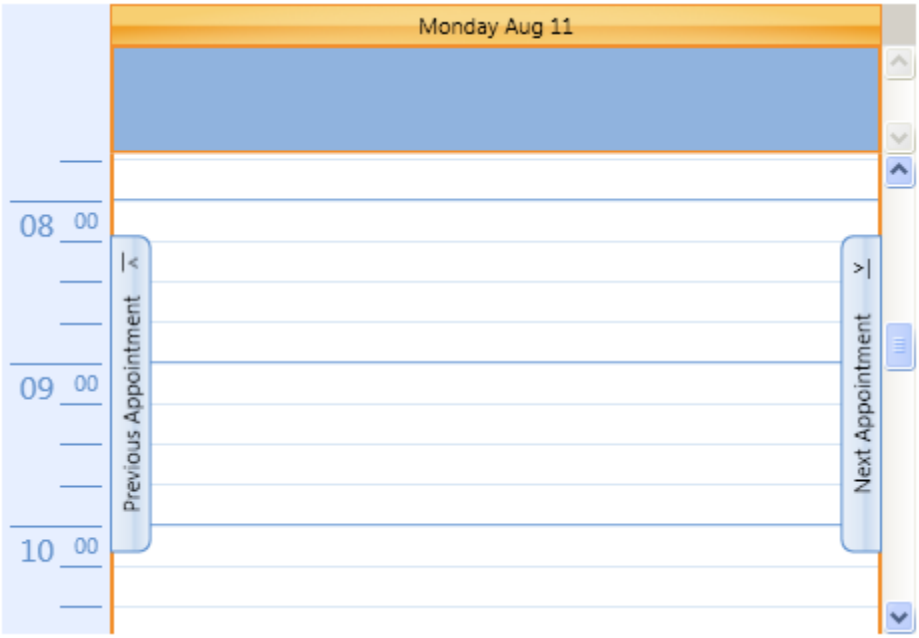
C1Scheduler Data View Keys and Appearance

Default view styles are defined with the following ComponentResourceKeys:

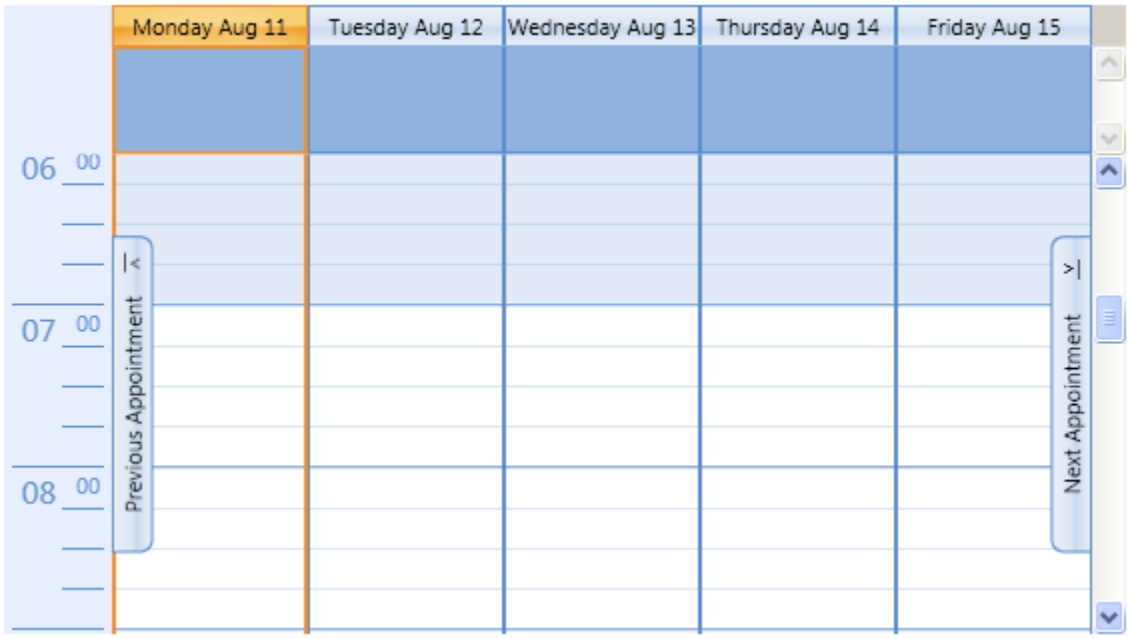
ComponentResourceKey	Description
x:Key="{ComponentResourceKey TypeInTargetAssembly={Type local:C1Scheduler}, ResourceId=OneDayStyle}" TargetType="{x:Type local:C1Scheduler}"	One Day View
x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=WorkingWeekStyle}" TargetType="{x:Type local:C1Scheduler}"	Work Week View
x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=WeekStyle}" TargetType="{x:Type local:C1Scheduler}"	Week View
x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceId=MonthStyle}" TargetType="{x:Type local:C1Scheduler}"	Month View

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

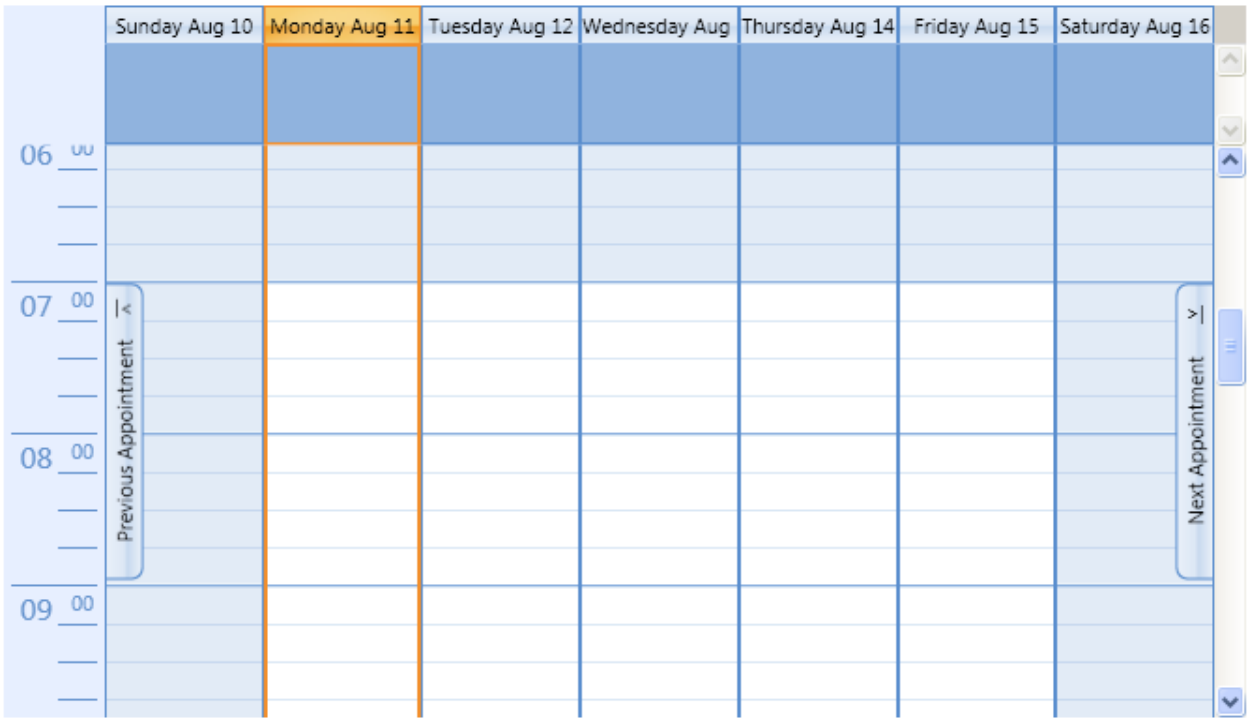
One Day View



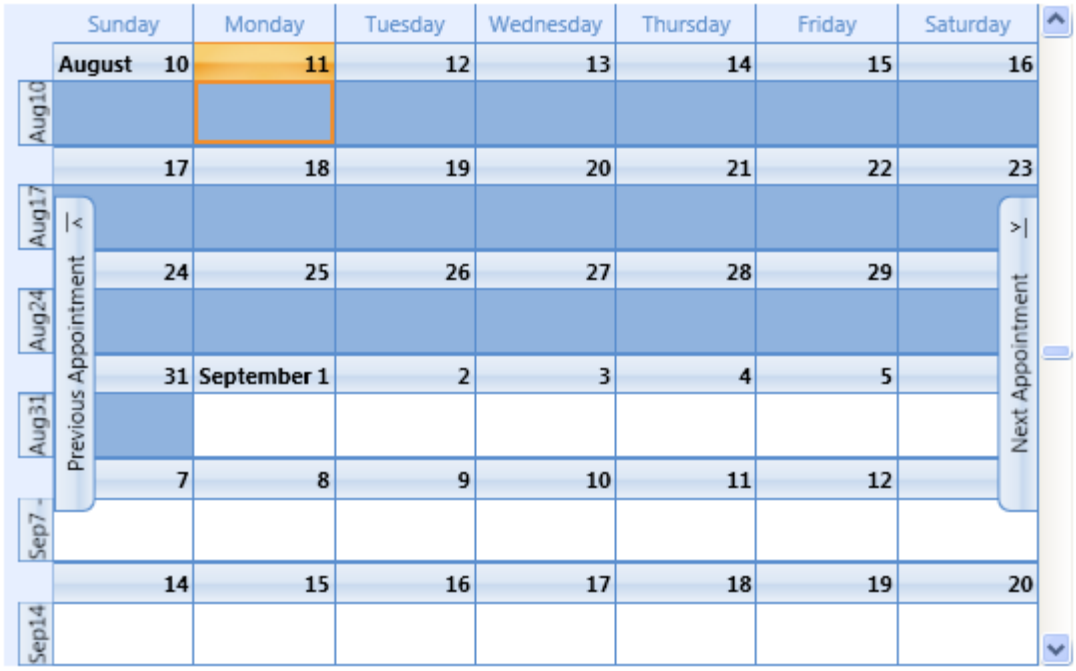
Working Week View



Week View



Month View



C1Scheduler Default Styles and Templates

Default styles are defined using styles and templates for smaller C1Scheduler parts. C1Scheduler uses these styles and templates as DynamicResources, so you can re-define any of them to customize default C1Scheduler views. The following table depicts the details of default C1Scheduler styles and templates along with their keys:

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.

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.
PART_C1Scheduler_Day_Style	Determines the style of the day group in a day view.

PART_C1Scheduler_Today_Style	Determines the style of the current day group in a day view.
C1Scheduler_WeekTab_Style	Determines the style of the week tab in a Month view.

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

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

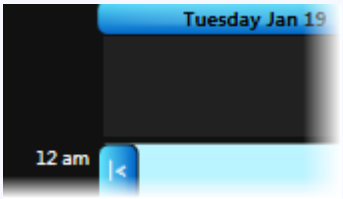
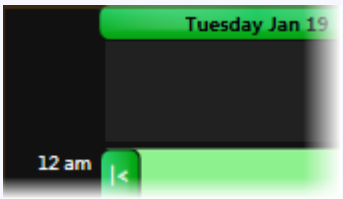
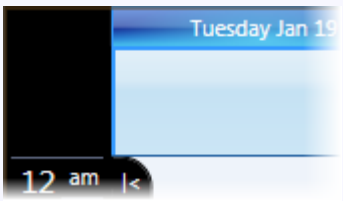
C1Scheduler's default interface includes some default DataTemplate objects. Custom user interfaces for this template can be provided as a DataTemplate object, which should be assigned to an appropriate C1Scheduler property. The default DataTemplates is accessible through ComponentResourceKeys. The following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

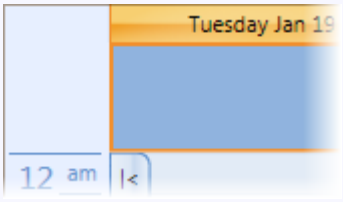
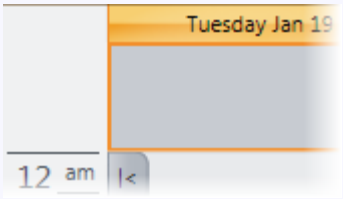
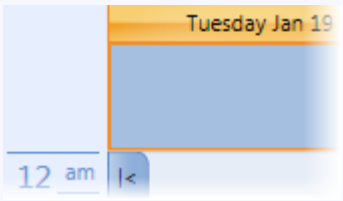
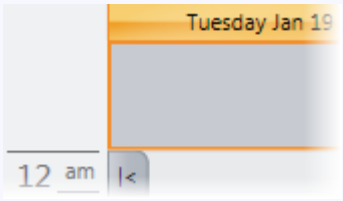
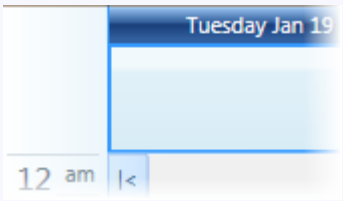
C1Scheduler Property	Default DataTemplate	Description
IntervalAppointmentTemplate	x:Key="{ComponentResourceKey TypeInTargetAssembly={x:Type local:C1Scheduler}, ResourceID=IntervalAppointmentTemplate}"	Appointment element template.

C1Scheduler 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\Studio for 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 PlayerTheme

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 Theme property.

To set the theme in Microsoft Blend, change the 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 Theme property and select **Reset**.
3. Click the Theme drop-down arrow again and choose one of the predefined themes.

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

```
<c1sched: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:

```
      <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.

Click here for a XAML example showing how to redefine theme brushes.

```
<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 -->
        <c1sched: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

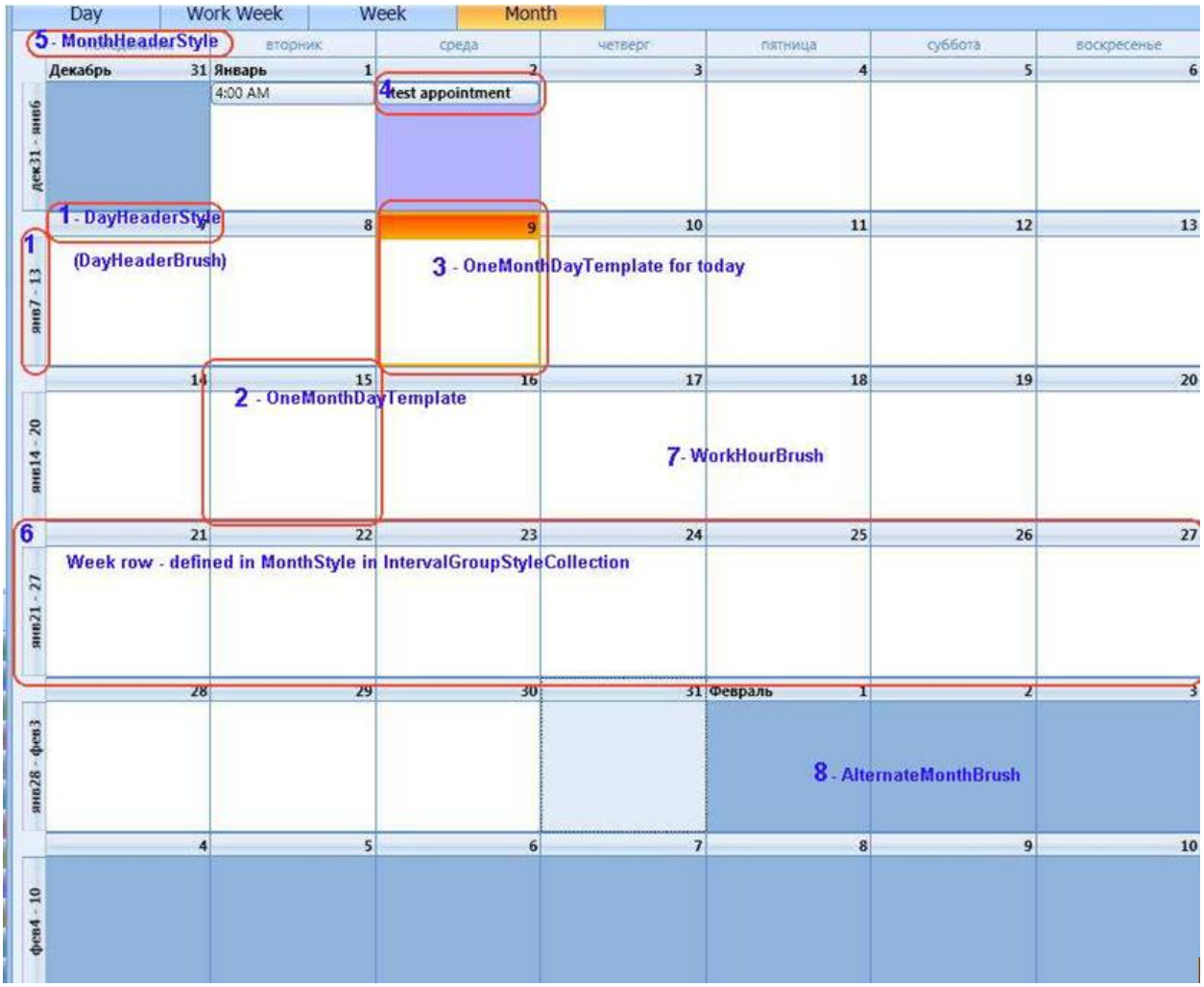
A theme is a ResourceDictionary containing a collection of different resources such as Brushes and Styles and DataTemplates that target the WPF controls. The C1Scheduler and C1Calendar controls are composed of many smaller parts, each of which is created dynamically at run-time. All of these parts are 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:

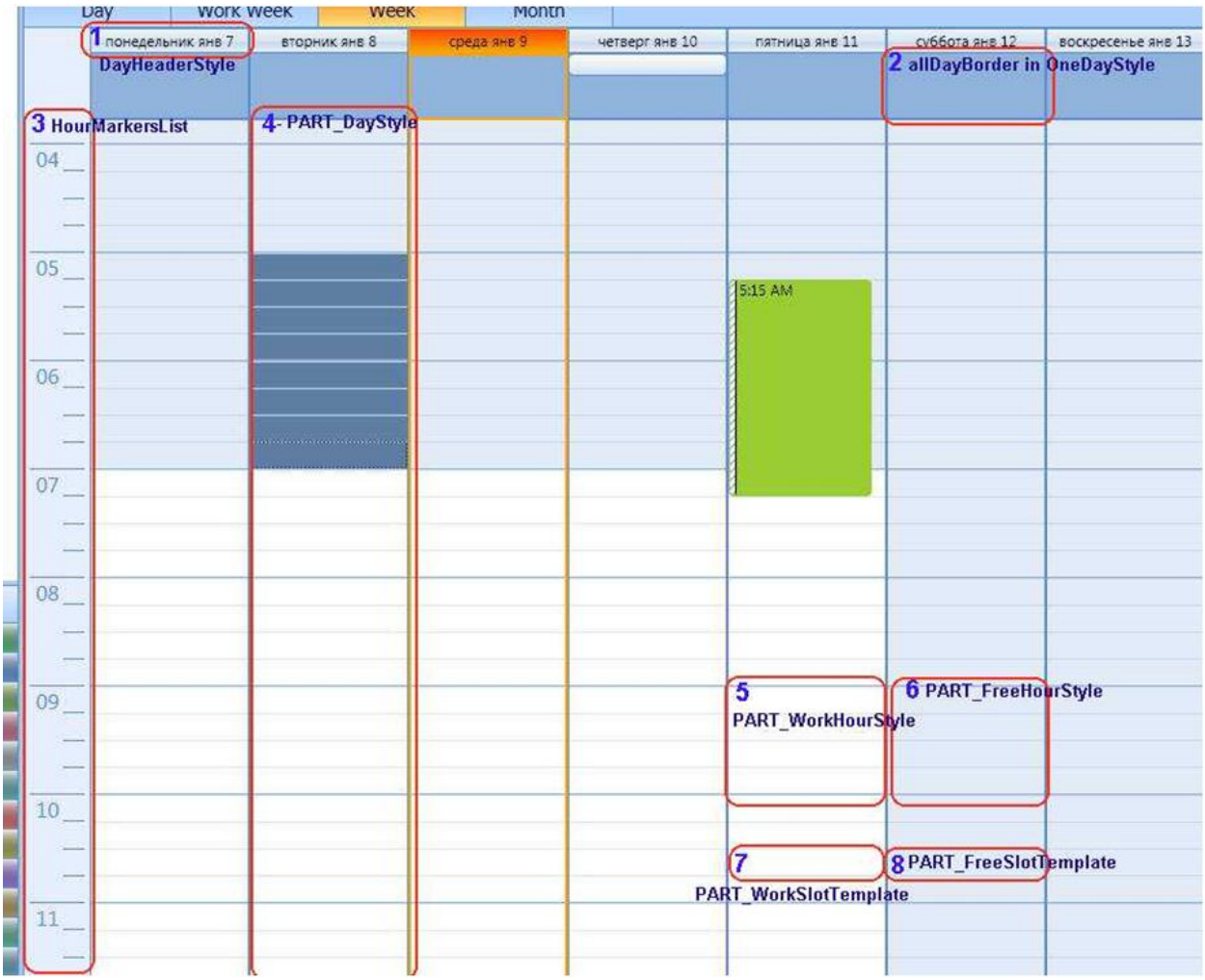
```
<!-- determines brush as frozen (for better performance) -->
<LinearGradientBrush x:Key="DayHeaderBrush" StartPoint="0,0" EndPoint="0,1"
PresentationOptions:Freeze="true">
   <GradientStop Color="#DDFFFFFF" Offset="0" />
   <GradientStop Color="#CAFFFFFF" Offset="0.5" />
   <GradientStop Color="#AAFFFFFF" 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. (page 13)
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:

```
<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/pre
sentation/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 ReourceDictionary.MergedDictionaries collection like in the following 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:

```
<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:

```
<!-- 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:
    ```
    <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 (page 13) 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:
    ```
    <Application.Resources>
          <ResourceDictionary
    Source="/C1SchedulerThemePack;component/themes/generic.xaml" />
    </Application.Resources>
    ```
7. Go back to the Window1.xaml file and change the Theme property of the C1Scheduler control to **MyTheme** using the following XAML. Delete grid tags.
    ```
    <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.

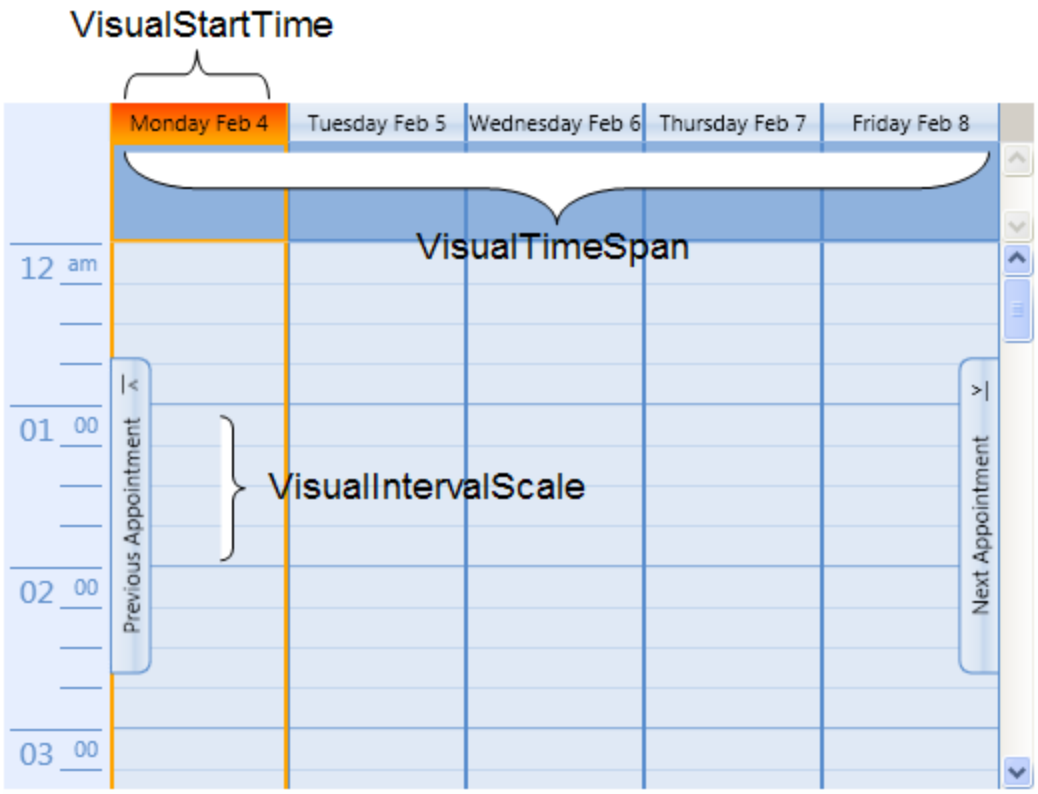
1. Create a **WPF Application (.exe)** in Blend.
2. If you want to redefine only part of the default C1Scheduler theme, copy all source XAML files with the theme definition into your application folder and add them to the project. Note that these files are installed with **ComponentOne Scheduler for WPF** to C:\Program Files\ComponentOne\Studio for WPF\C1WPFNewSchedule\XAML\themes, by default. They may be in a different location if you installed ComponentOne Studio for WPF 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 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.

Customizing the User Interface

Scheduler for WPF 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 VisualStartTime sets the first date to appear in the schedule. The VisualTimeSpan property defines the amount of time, or in most cases, the number of days shown in a view. The VisualIntervalScale property breaks the VisualTimeSpan even further into increments.

Using the *Working Week View* view style as an example, the VisualStartTime is the first Monday of the current week, by default. The VisualTimeSpan is 5:00:00:00, or five days, Monday through Friday which is a typical work week. The VisualIntervalScale is 00:15:00 which means that each hour of the schedule is shown broken down into 15 minute intervals.



Note: The VisualStartTime property value is determined automatically by the VisualTimeSpan and SelectedDateTime properties, so it should not be set in most cases; however, the VisualTimeSpan and 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 VisualStartTime, VisualTimeSpan, and VisualIntervalScale properties, the VisualIntervalCollection is filled with VisualInterval objects, each defining a period of time of the 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 VisualIntervalTemplate property should be used. The VisualIntervalPanel property defines a Panel where the VisualInterval's items are placed.

To provide a group view for the VisualIntervalCollection, the VisualIntervalsView property should be used, which is a **System.Windows.Data.ListCollectionView** 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:

```
C1Scheduler1.ChangeStyle(C1SchedulerResources.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:

```
<!-- Group descriptions -->
<Setter Property="VisualIntervalGroupDescriptions">
      <Setter.Value>
            <local:IntervalGroupDescriptionCollection>
                  <!-- Group Level1 - grouped by days -->
                  <local:VisualIntervalGroupDescription
PropertyNames="StartTime.Day" />
                  <!-- Group Level2 - grouped by hours -->
                  <local:VisualIntervalGroupDescription
PropertyNames="StartTime.Hour" />
            </local:IntervalGroupDescriptionCollection>
      </Setter.Value>
</Setter>

<!-- Group styles definition -->
<Setter Property="local:VisualIntervalGroupStyles">
      <Setter.Value>
            <local:IntervalGroupStyleCollection>
                  <!-- Group Level1 - grouped by days -->
                  <GroupStyle ContainerStyleSelector="{StaticResource
DayGroupStyleSelector}">
                        <GroupStyle.HeaderTemplate>
                              <DataTemplate>
                                    <Border Visibility="Collapsed" />
                                    <!-- Header is drawn separately -->
                              </DataTemplate>
```

```
                        </GroupStyle.HeaderTemplate>
                        <!-- The group is a single row of days -->
                        <GroupStyle.Panel>
                              <ItemsPanelTemplate>
                                    <UniformGrid SnapsToDevicePixels="True"
Rows="1" />
                              </ItemsPanelTemplate>
                        </GroupStyle.Panel>
                  </GroupStyle>
                  <!-- Group Level2 - grouped by hours -->
                  <GroupStyle ContainerStyleSelector="{StaticResource
TimeSlotGroupStyleSelector}">
                        <GroupStyle.HeaderTemplate>
                              <DataTemplate>
                                    <Border Visibility="Collapsed" />
                              </DataTemplate>
                        </GroupStyle.HeaderTemplate>
                        <!-- The group is a single column of 24 hours -->
                        <GroupStyle.Panel>
                              <ItemsPanelTemplate>
                                    <UniformGrid Rows="24" />
                              </ItemsPanelTemplate>
                        </GroupStyle.Panel>
                  </GroupStyle>
            </local: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 are 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 VisualIntervalPanel property.

To define the VisualInterval representation, either set the VisualIntervalTemplate property or use the TimeSlotTemplateSelector.

Simplifying User Interface Creation

In order to simplify the building of a user interface based on Scheduler for WPF'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, allow 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 an 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 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 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 IntervalAppointmentTemplate property.

The AppointmentsCoverPane control provides functionality for an arbitrary UI representing VisualIntervals. To make this possible, each element that can be treated as a VisualInterval UI representative, usually an outer (root) element in the VisualIntervalTemplate definition, and must have the attached 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 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:

```
<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 Visual Intervals

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

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:
   - 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:
   - 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:
   - 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 VisualIntervalTemplate property.

Enhancing the User Interface with Method Calls

Just getting or setting properties is not enough to provide a fully-functional user interface. Often method calls are necessary. The Windows Presentation Foundation (WPF) provides the **ObjectDataProvider** class, which is intended to allow you to perform method or constructor calls declaratively in XAML. However, the **ObjectDataProvider** properties that define what method with what parameters on which object to call are not DependencyProperties, so binding to them, if possible, is difficult. Trigger Setters can't be used to define these properties. Also, there is no convenient way to control when the call should be performed; it happens each time one of the properties has been changed.

The **MethodCaller** class provides similar behavior to WPF's **ObjectDataProvider**, but its properties are DependencyProperties, allowing you to conveniently bind to them and use them in Trigger Setters. Because it's derived from the **FrameworkElement** class, like the PropertyBridge class, the **MethodCaller** class can be placed in the visual tree which allows bindings to work properly.

In addition to the features provided by **ObjectDataProvider**, the **MethodCaller** class provides the following:

- MethodParameters(), MethodParameters1, MethodParameters2, and so on are dependency properties, instead of a collection of parameters, which allows you to define method parameters via binding.
- The Boolean **MethodCaller.Call** dependency property. A method or a constructor call is performed when this property changes its value from **False** to **True**. It can be set from within a trigger or at a fully controlled time when the method is called, for example, when bound to the **IsPressed** property of **Button**.
- The Boolean **MethodCaller.PermanentCall** property. When set to **True**, **MethodCaller.PermanentCall** forces the **MethodCaller** class to work in the same manner as the **ObjectDataProvider** class. In other words, a call is performed each time some of its properties have been changed.

Scheduler 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 **ButtonBase** or **InputBinding** 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 VisualStartTime property value on the amount specified in the SmallStartTimeChange property.
DecrementStartTimeLargeCommand	Decrements the VisualStartTime property value on the amount specified in the LargeStartTimeChange property.
IncrementStartTimeSmallCommand	Increments the VisualStartTime property value on the amount

	specified in the SmallStartTimeChange property.
IncrementStartTimeLargeCommand	Increments the VisualStartTime property value on the amount specified in the LargeStartTimeChange property.
SetRelativeStartTimeCommand	Sets the 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	Makes a nearest appointment before the SelectedDateTime visible in the control UI.
NavigateToNextAppointmentCommand	Makes a nearest appointment after the SelectedDateTime 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: 6. The master list to choose from. For example Resources. 7. The resulting list where to put selected items. For example, the Appointment.Resources list. 8. The System.Type value specifying the type of items in both lists. 9. The reference to the owning window if any. 10. 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	
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.

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 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 DataMember property is assigned to the **Appointments** table of a database. The DataSource is set to the data set resource in the project:

```
<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

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:
    1. Select File | New Project.
    2. 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#**.
    3. Select **WPF Application** from the *Templates* pane.
    4. 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:
    1. Select Project | Add Reference.
    2. Browse to find the location of the assembly file. When **Scheduler for WPF** is installed, this file is installed to C:\Program Files\ComponentOne\Studio for WPF\bin, by default. Note that the location may be different if you performed a custom install.
    3. Add the following using statements to the page:
    ```
    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:
    1. Select **Data | Add New Data Source**. The **Data Source Configuration Wizard** appears.
    2. Select **Database** and click **Next**.
    3. 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.

4. 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.
5. Click **Next** to save the connection string.
6. 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.

```
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.
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:

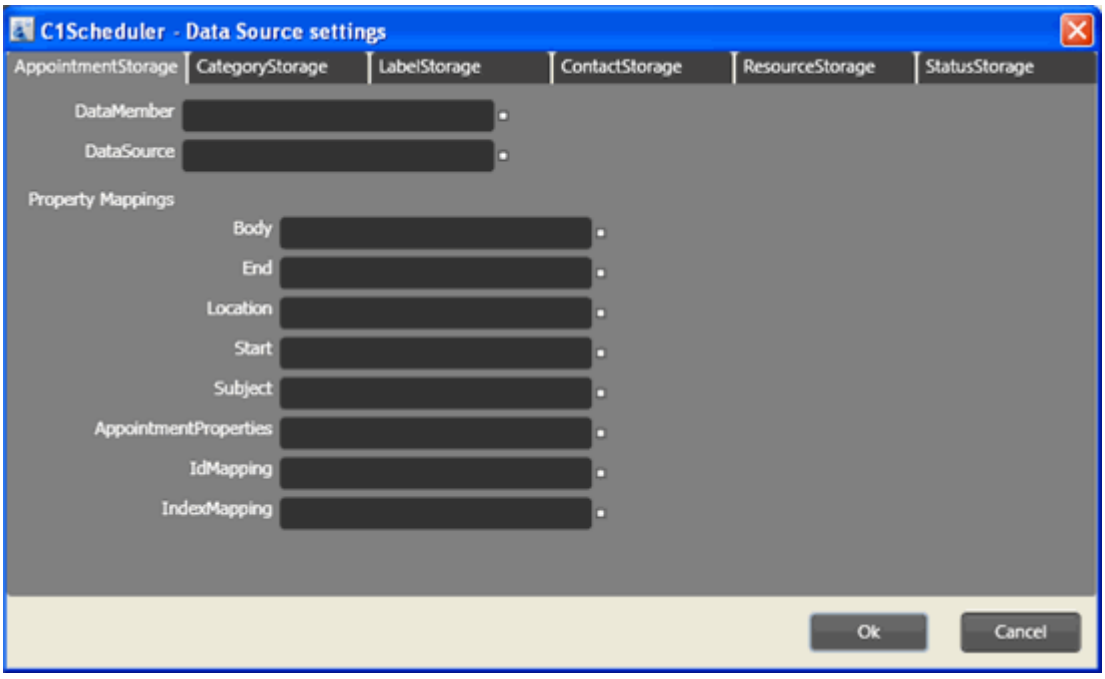
```
<Page
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    x:Class="WpfApplication1.Page1"
      xmlns:local="clr-namespace:WpfApplication1"
    Title="Page1"
xmlns:c1sched="http://schemas.componentone.com/wpf/Schedule">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

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

```
<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:

```
      <c1sched:C1Scheduler Margin="0,0,-80,-80" Theme="{DynamicResource
{ComponentResourceKey ResourceId=Office2007.Blue, TypeInTargetAssembly={x:Type
c1sched:C1Scheduler}}}">
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataMember" Value="Appointments"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.DataSource" Value="{DynamicResource
dataSet}"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Body.MappingName"
Value="Body"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.End.MappingName"
Value="End"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Location.MappingName"
Value="Location"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Start.MappingName"
Value="Start"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.Subject.MappingName"
Value="Subject"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.AppointmentProperties.Mappi
ngName" Value="Properties"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.IdMapping.MappingName"
Value="Id"/>
      <c1sched:NestedPropertySetter
PropertyName="DataStorage.AppointmentStorage.Mappings.IndexMapping.MappingName"
Value="N/A"/>
      </c1sched:C1Scheduler>
```

Data Binding Using the PropertyBridge Class

The PropertyBridge class exposes two dependency properties, Source and 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 Source property with a TwoWay binding having one non-dependency property as a source, and assign the 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 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 Target property of PropertyBridge – this value will be assigned to the non-dependency property which is bound to Source.
- **Many-to-many binding:**
  Assign Source and 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 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 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.

Adding Localized Resources to a Project

ComponentOne Scheduler for WPF projects can be localized; the **C1.WPF.Schedule** assembly contains resources for English and Japanese cultures.

All C1Scheduler default templates honor the Culture property and use the corresponding string resources, if any.

To add localized resources to your project:

1. Create a copy of the .resx files included with **ComponentOne Scheduler for WPF**. These resource files are installed in **C:\Program Files\ComponentOne\Studio for WPF\C1WPFNewSchedule\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-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 solution. After rebuilding the project, satellite assemblies for each culture will be created in subfolders of the output directory. Ship these assemblies (do NOT change directory tree) along with your application.

> **Note:** The C1Scheduler control uses default English resources embedded into the C1.WPF.C1Schedule.dll as a resource fallback. If you are localizing the C1Scheduler control in a custom assembly within your application, you can change this behavior by setting your assembly to contain default resources. To do this, set the C1.WPF.Localization.C1Localizer.DefaultAssembly property|topic=DefaultAssembly property to the custom assembly name. Do not include the extension, version number, and so on.

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

Resource File	Description
Categories.resx	Contains localized names for default appointment categories.
EditAppointment.resx	Contains localized strings for default EditAppointmentTemplate.
EditRecurrence.resx	Contains localized strings for default EditRecurrenceTemplate.
Exceptions.resx	Contains localized error messages which might be shown to end user.
Labels.resx	Contains localized names for default appointment labels.
MiscStrings.resx	Contains localized strings used for describing recurrence pattern.
RecChoice.resx	Contains localized strings for Open/Remove recurrence dialogs.
SelectFromListScene.resx	Contains localized strings for SelectFromListScene control.
ShowReminders.resx	Contains localized strings for default ShowRemindersTemplate.
Statuses.resx	Contains localized names for default availability statuses.
TimeStrings.resx	Contains localized strings for the C1TimeSpanPicker control.
ValidationErrors.resx	Contains localized strings for validation errors.

> **Note:** You can localize only the part of these files. For example, if you use custom EditAppointmentTemplate, you do not need resources from the EditAppointment.resx.

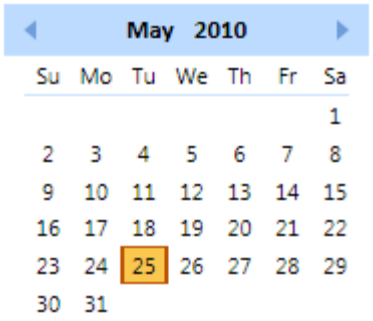
Known Issues

The Edit Template Copy feature of Microsoft Expression Blend does not create an exact copy of the source XAML and omits custom MarkupExtensions. Blend adds strings for the default culture into the template copy. Compare resulting templates with the source xaml and correct them if you want to use the localization feature in your templates.

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
SelectedDate DateTime SelectedDate {get; set;}	Defines a current date that is selected in calendar.
Year int Year {get; set;}	Defines a year component of the C1.WPF.Schedule.CalendarBase.SelectedDate property.
Month int Month {get; set;}	Defines a month that is represented by a calendar.
MaxDate DateTime MaxDate {get; set;}	Gets or sets the maximum allowable date. The default value is 12/31/9998.
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
CalendarHelper, public C1.WPF.Schedule.CalendarHelper CalendarHelper {get; set;}	Provides calendar-dependant properties.
MaxSelectionCount, public int MaxSelectionCount {get; set;}	Defines the maximum number of days that can be selected in the control.
SelectedDates, public C1.WPF.Schedule.DateList SelectedDates {get; set;}	The list of selected dates.
BoldedDates, public C1.WPF.Schedule.DateList BoldedDates {get; set;}	The list of bolded dates.
DaysPanel, public System.Windows.Controls.ItemsPanelT	An ItemsPanelTemplate that defines the panel that lays out elements representing days of a month. By default the

emplate DaysPanel {get; set;}	AutoDistributionGrid panel with 7 columns and 6 rows is used.
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.
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.
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.
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.
MonthFullName, public string MonthFullName { get; }	Gets a full name of a month currently represented by calendar taking into account current culture.
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
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:

11. A pane that represents a list of the month's days
12. A pane that represents a list of days of the week names
13. 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 Year and Month properties. An actual UI of these DaySlotPresenter elements is defined in the DaySlotTemplate property. Those elements become children of a panel whose UI is defined in the 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 Empty - indicates whether a slot represents a day or it's an empty one
- DateTime Date - gets a date represented by the DaySlot or a null value if the DaySlot is empty
- DayOfWeek DayOfWeek - gets the day of the week that this slot corresponds to
- bool IsWeekEnd - indicates whether this day is a weekend
- bool IsSelected - indicates whether this day slot is currently selected
- bool IsAdjacent - indicates whether the DaySlot represents a day from an adjacent month, but not of the month currently represented by the C1Calendar
- bool 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 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 DayOfWeekSlotPresenter objects as children of a panel whose template is defined in the C1Calendar DaysOfWeekPanel. Each DayOfWeekSlotPresenter object represents a single day of the week. A UI of each DayOfWeekSlotPresenter is defined in the C1Calendar 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 DayOfWeek – gets the day of the week represented by the DayOfWeekSlot
- string DayFullName – the culture-specific full name of a day;
- string DayShortName – the culture-specific abbreviated name of a day;
- string DayShortestName – the culture specific shortest name for the DayOfWeek;
- bool 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

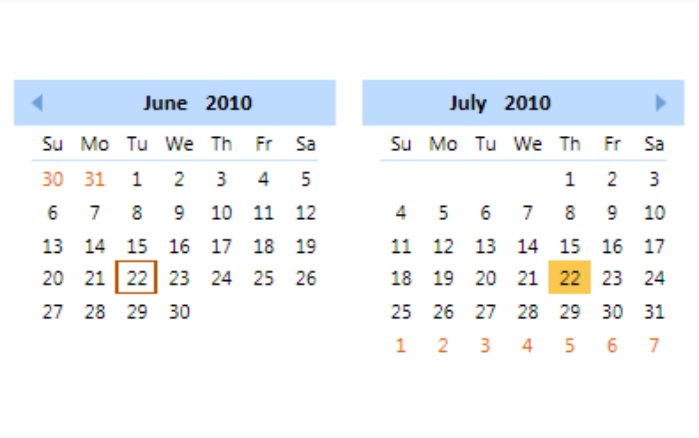
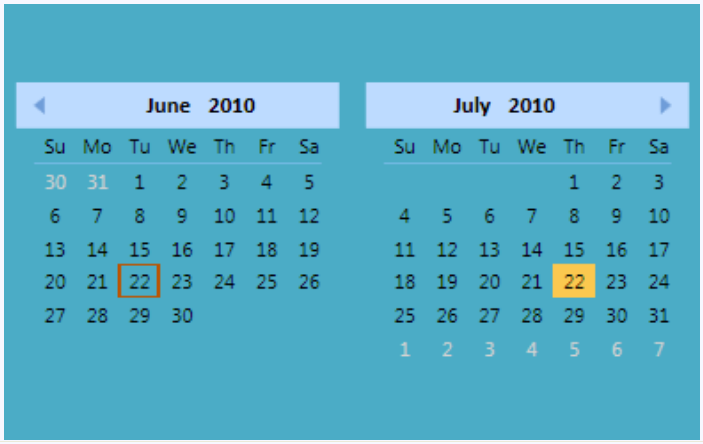
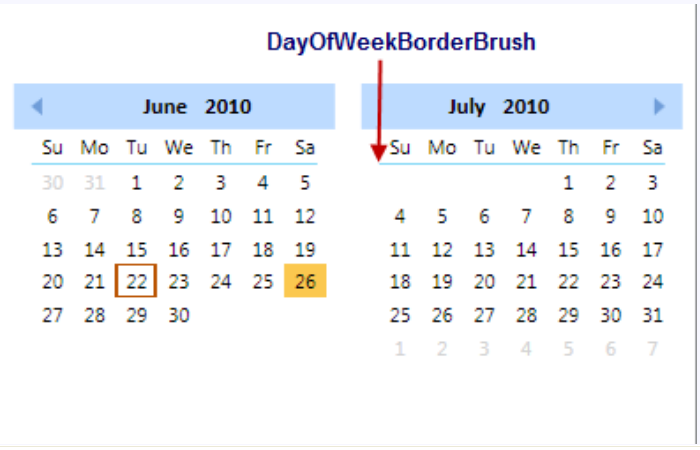
ComponentOne Scheduler for WPF includes several properties that allow you to customize the appearance of the calendar control. You can change the color, border, and height of the control. The following topics describe some of these appearance properties:

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.

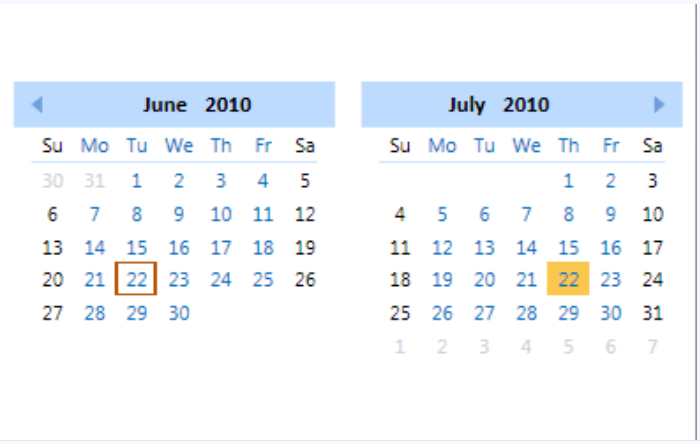
Property	Description
AdjacentMonthDayBrush	Gets or sets a Brush object used to display adjacent month days. This is a dependency property.
Background	Gets or sets a brush that describes the background of a control. This is a dependency property.
DaysOfWeekBorderBrush	Gets or sets a Brush object used 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.
MonthHeaderBackground	Gets or sets a Brush object used to color month header. This is a dependency property.
MonthHeaderForeground	Gets or sets a Brush object used to color month header text. This is a dependency property.
NavigationButtonBrush	Get or sets a Brush used to color navigation buttons. This is a dependency property.
SelectedDayBrush	Gets or sets a Brush object used to highlight selected dates. This is a dependency property.
TodayBrush	Gets or sets a Brush object used to highlight current date. This is a dependency property.
WeekendBrush	Gets or sets a Brush object used to display weekends. This is a dependency property.

Property	Example

AdjacentMonthDayBrush	
Background	
DaysOfWeekBorderBrush	

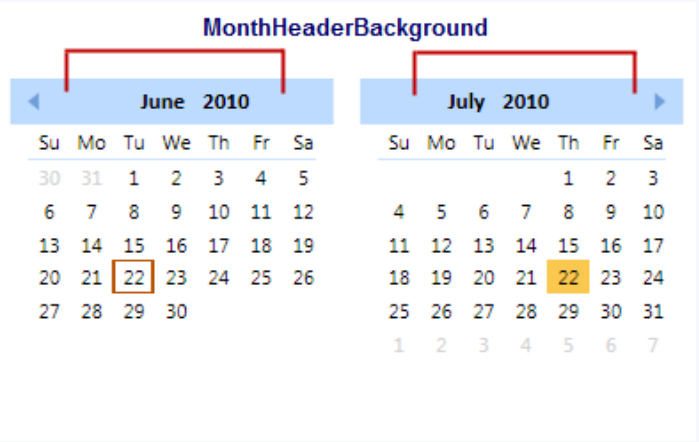
Foreground

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



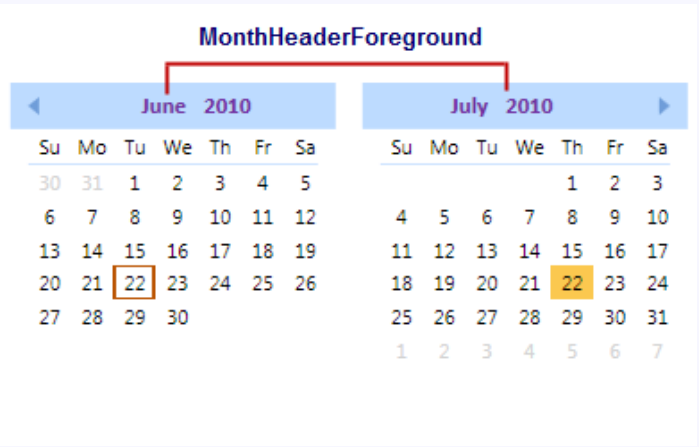
MonthHeaderBackground

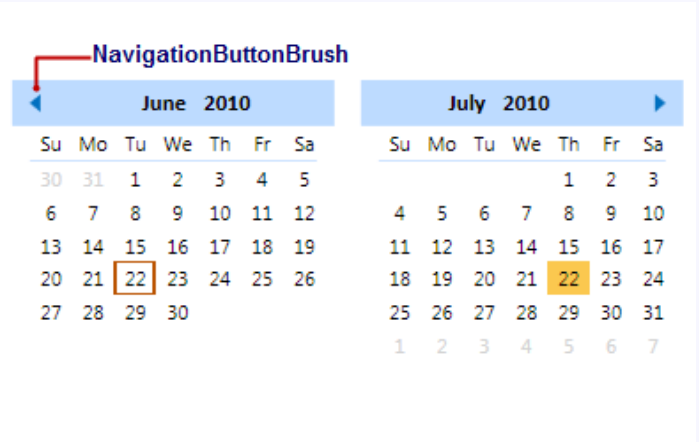
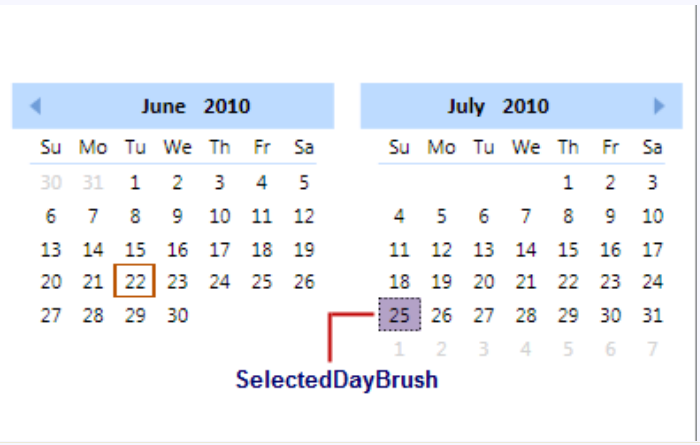
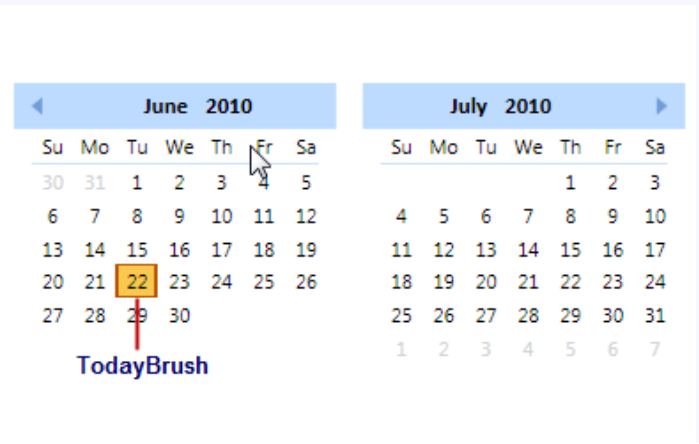
This property affects the background color for the month header.

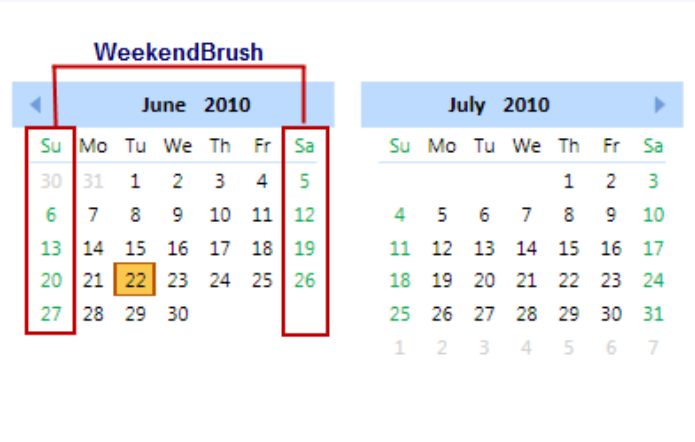


MonthHeaderForeground

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



NavigationButtonBrush	
SelectedDayBrush	
TodayBrush	

WeekendBrush	

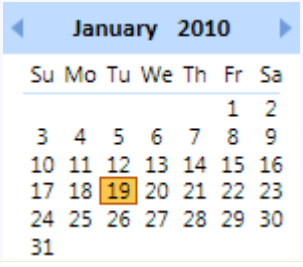
Text Properties

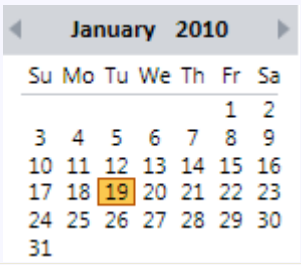
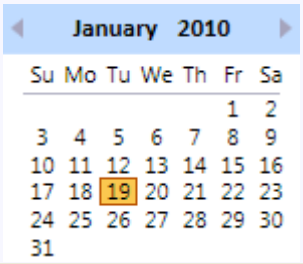
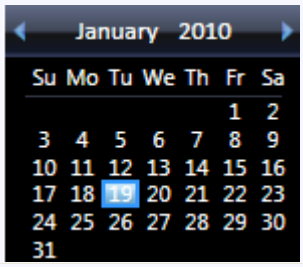
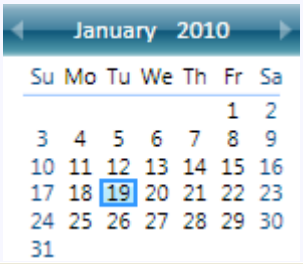
The following properties let you customize the appearance of text in the calendar control.

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.
MonthHeaderFontFamily	Gets or sets a FontFamily object used to display month header text. This is a dependency property.
MonthHeaderFontSize	Gets or sets a Double value determining month header font size. This is a dependency property.
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\Studio for WPF\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:

14. Right-click the control.
15. 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 Theme property.

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

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

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

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

18. 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.
19. To define the Days pane UI, assign templates to the following properties:
    - C1Calendar DaysPanel – defines a panel for the layout of day items;
    - C1Calendar 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 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.
20. To define the Days of Week pane UI, assign templates to the following properties:
    - C1Calendar DaysOfWeekPanel – defines a panel for the layout of days of week items;
    - C1Calendar 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 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

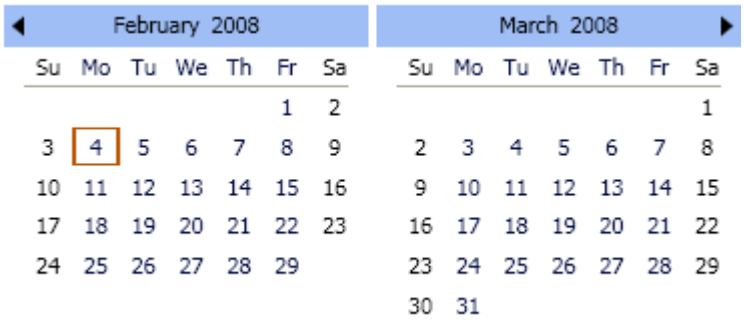
ComponentOne Calendar for WPF includes several properties that allow you to customize the layout of the calendar control. You can change the width, height, alignment, and the amount of calendar months displayed of the control. The following properties let you customize the control's layout:

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.
MonthCount	Gets or sets a number of months currently represented by the calendar. The default value is 1. This is a dependency property.
MonthHeight	Determines a height of each month slot of the calendar.
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 MonthCount – the number of months currently represented by the calendar;
- Style MonthCalendarStyle – a Style applied to each child C1Calendar representing a single month;
- Style MonthSlotStyle – a Style for C1CalendarPresenter elements which are the root elements of a visual tree representing a single month;
- ItemsPanelTemplate MonthsPanel – an ItemsPanelTemplate that defines the panel that lays out elements representing separate months.

To provide a custom look for a C1Calendar:

21. 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.
22. To define the Months pane UI, assign a template to the MonthsPanel property. This template will define a panel that lays out month items.
23. To define a single month UI, assign the Style to the 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

Calendar for WPF 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 BoldedDates and 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.

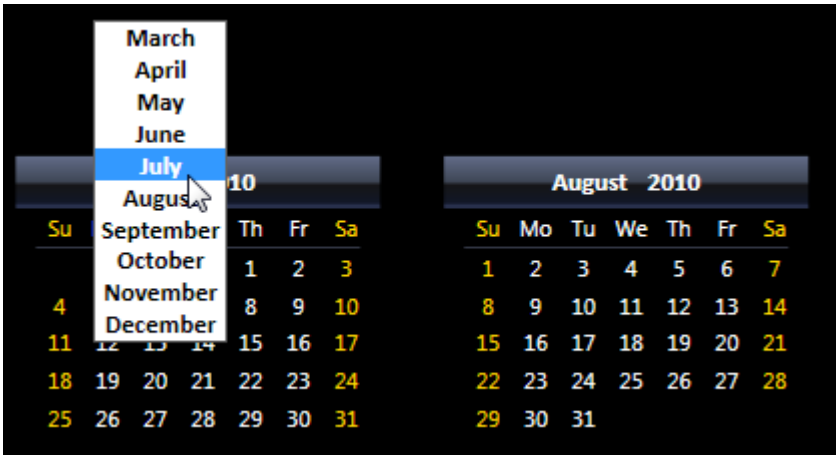
Previous and Next Navigation Buttons

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.

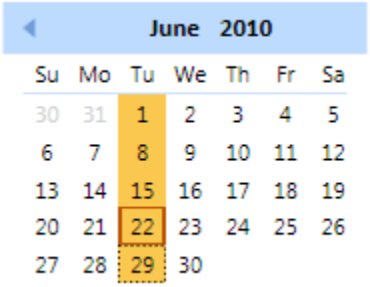
You can click the calendar month once to view calendar months. When you click once on the month name in the calendar title a popup listbox will appear with the names of the calendar months. Select the month you wish to navigate to and the calendar month will change from the current month to the selected 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 MaxSelectionCount property. At run time depending on the value of the 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 (page 106).

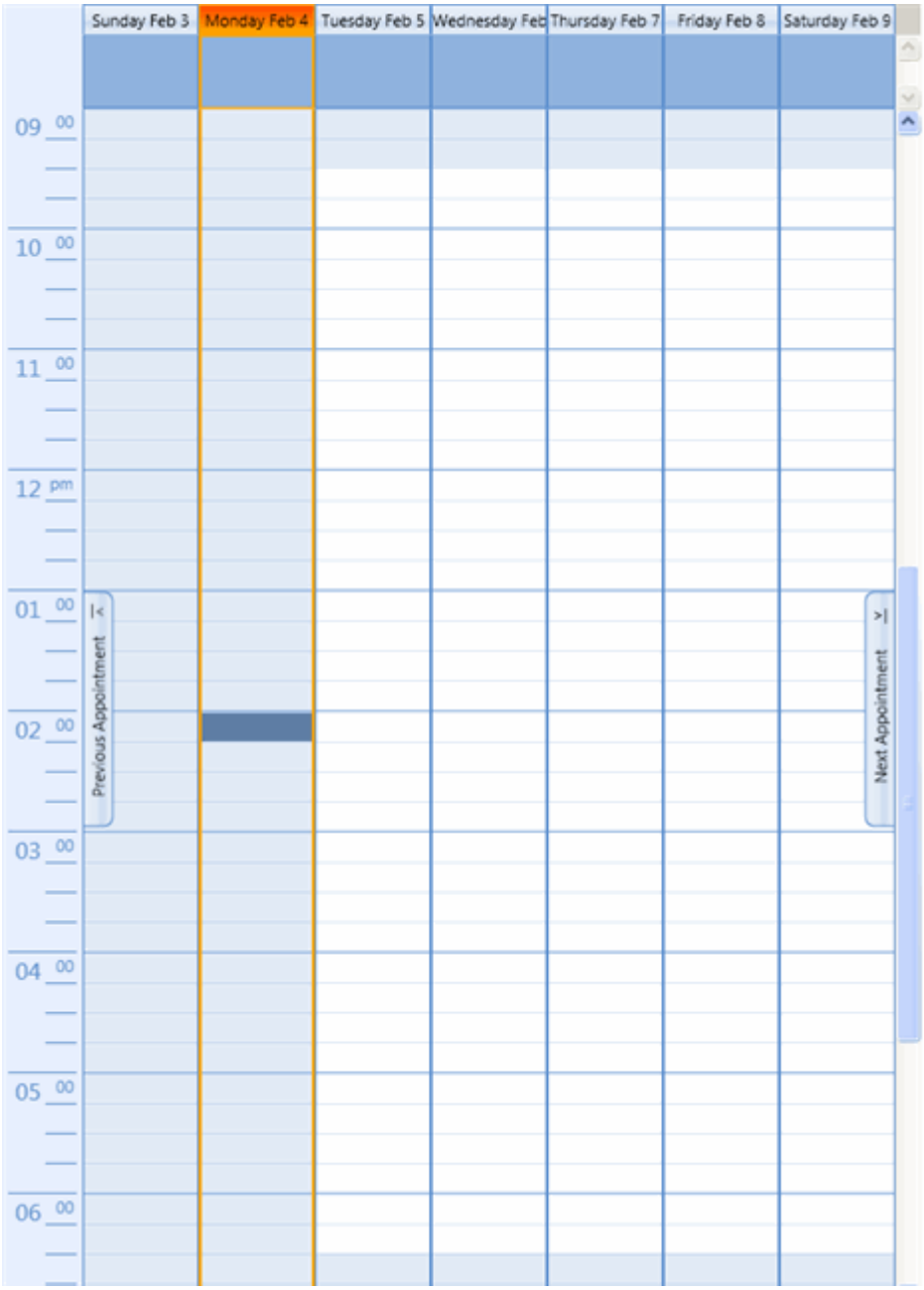
You can specify the maximum and minimum date for date selection on the calendar through the MaxDate and MinDate properties. By default the minimum is 1/1/1753 and the maximum is 12/31/9998. When the maximum and minimum dates are applied, the dates past these values become unselectable. As a result, the 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

```
<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

```
<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 CalendarHelper properties for one control and use them in several controls. This can be done by binding the CalendarHelper properties of corresponding controls to each other. For example, suppose you have a C1Scheduler control with the previously specified CalendarHelper properties and a C1Calendar control with no

properties specified yet in your window. Set the C1Calendar CalendarHelper properties as in the following XAML:

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

The C1Calendar will display all of the 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.

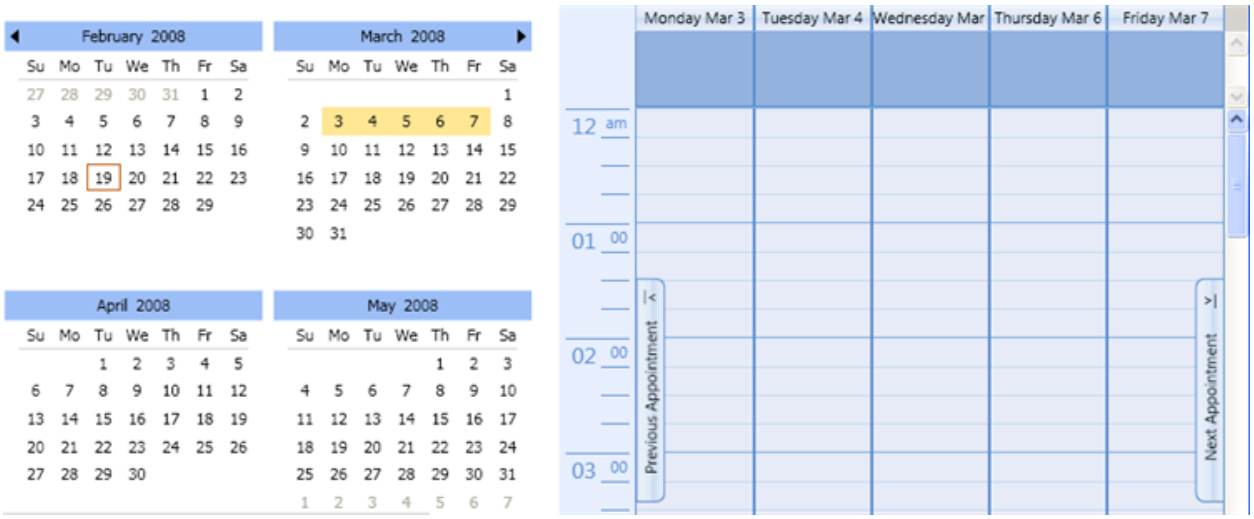
```
<Window
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:System="clr-namespace:System;assembly=mscorlib"
      x:Class="UntitledProject1.Window1"
      x:Name="Window"
      Title="Window1"
      Width="924"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

<Grid>
  <Grid.ColumnDefinitions>
    <ColumnDefinition/>
    <ColumnDefinition/>
  </Grid.ColumnDefinitions>
  <c1sched: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}}">
      <c1sched:C1Scheduler.CalendarHelper>
            <c1sched:CalendarHelper WeekStart="Sunday" EndDayTime="18:20:00"
StartDayTime="09:20:00"
                  WorkDays="Tuesday,Wednesday,Friday,Saturday">
                  <c1sched:CalendarHelper.Holidays>
                        <System:DateTime>2007-11-02</System:DateTime>
                  </c1sched:CalendarHelper.Holidays>
            </c1sched:CalendarHelper>
      </c1sched:C1Scheduler.CalendarHelper>
  </c1sched:C1Scheduler>

  <c1sched: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" >
     </c1sched:C1MultiMonthCalendar>
</Grid>
```

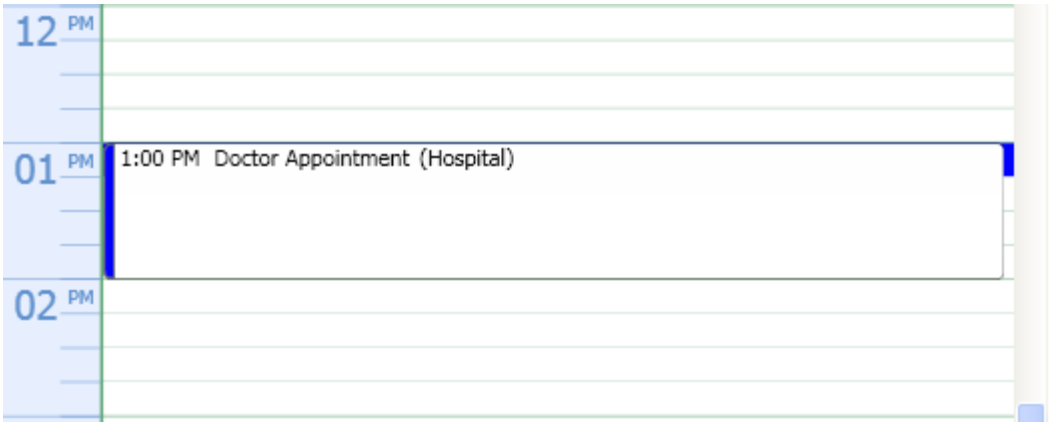
```
</Window>
```

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



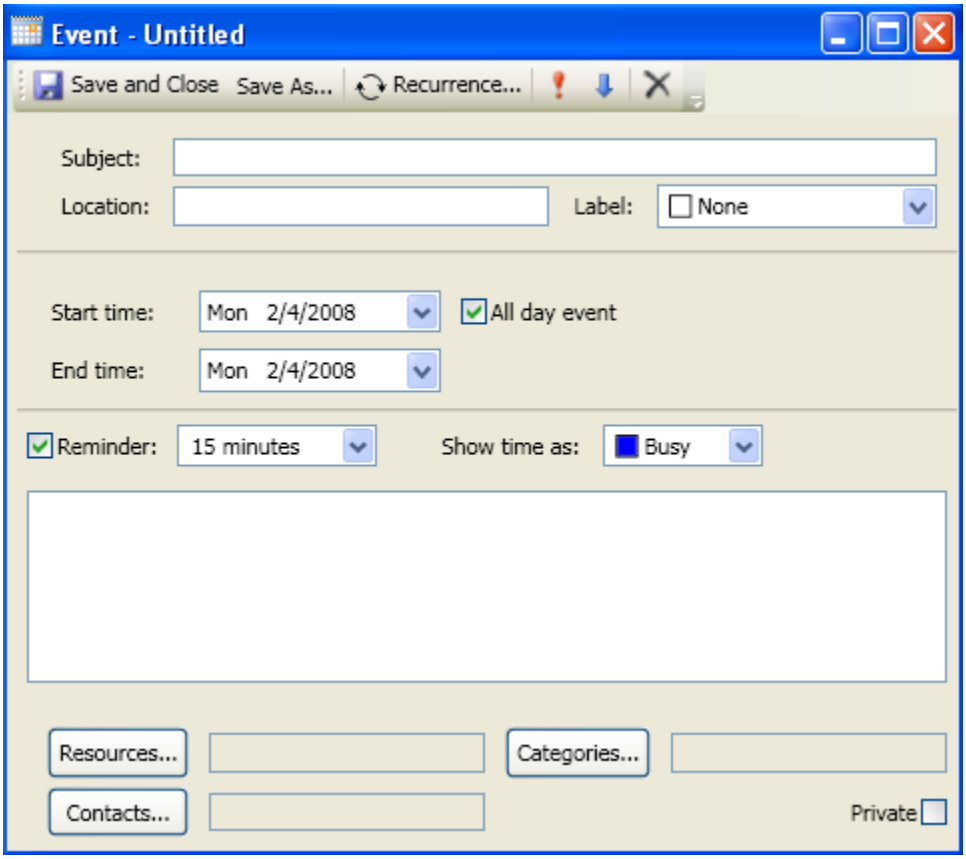
Appointments

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.

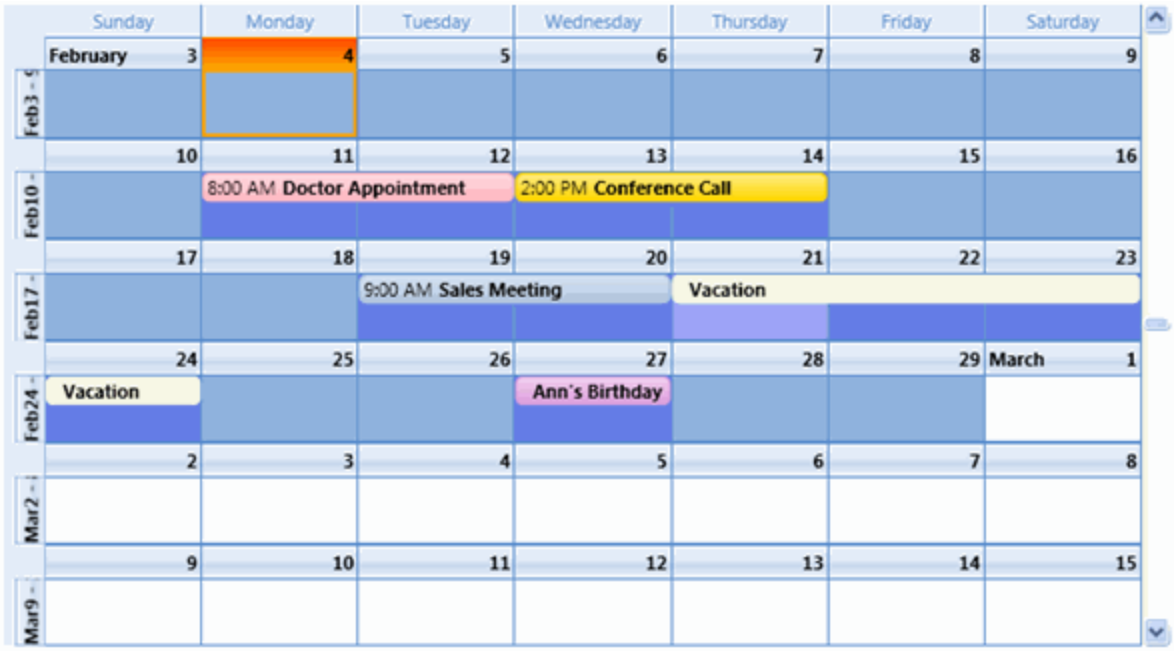
The **Appointment** dialog box is used to schedule new appointments, allowing you to set a subject, location, label, start and end time, reminder, availability status, and whether the appointment is an all day event and recurring over a specified period of time. You can also specify any resources, categories, 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**. The color of the label is visible in every data view in the C1Scheduler control.



Predefined Labels

The predefined labels include the following:

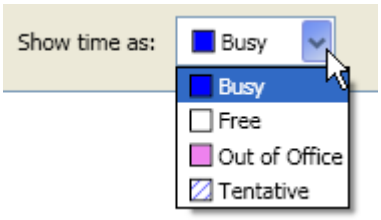
Label	Color	Index
None		0
Important		1
Business		2
Personal		3
Vacation		4
Deadline		5
Must Attend		6
Travel Required		7
Needs Preparation		8
Birthday		9
Anniversary		10
Phone Call		11

Availability Status

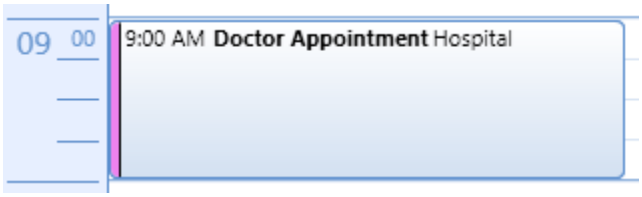
The availability status lets everyone know if you will be available at the time when an appointment is scheduled. There are four predefined availability statuses available in **Scheduler for WPF**: *Busy*, *Free*, *Out of Office*, and *Tentative*. The status is specified by the following colors:

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.

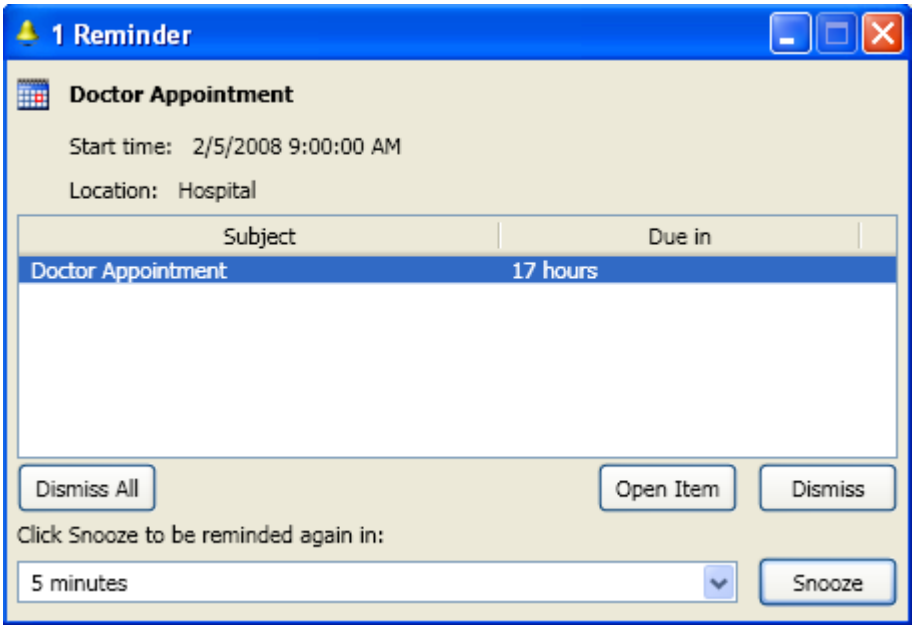


The color of the availability status is only visible in the **OneDayStyle** and **WorkingWeekStyle** views of a schedule. For appointments, the color appears in the area to the left of the appointment. Note that if you select the appointment, it will appear highlighted in the status color.

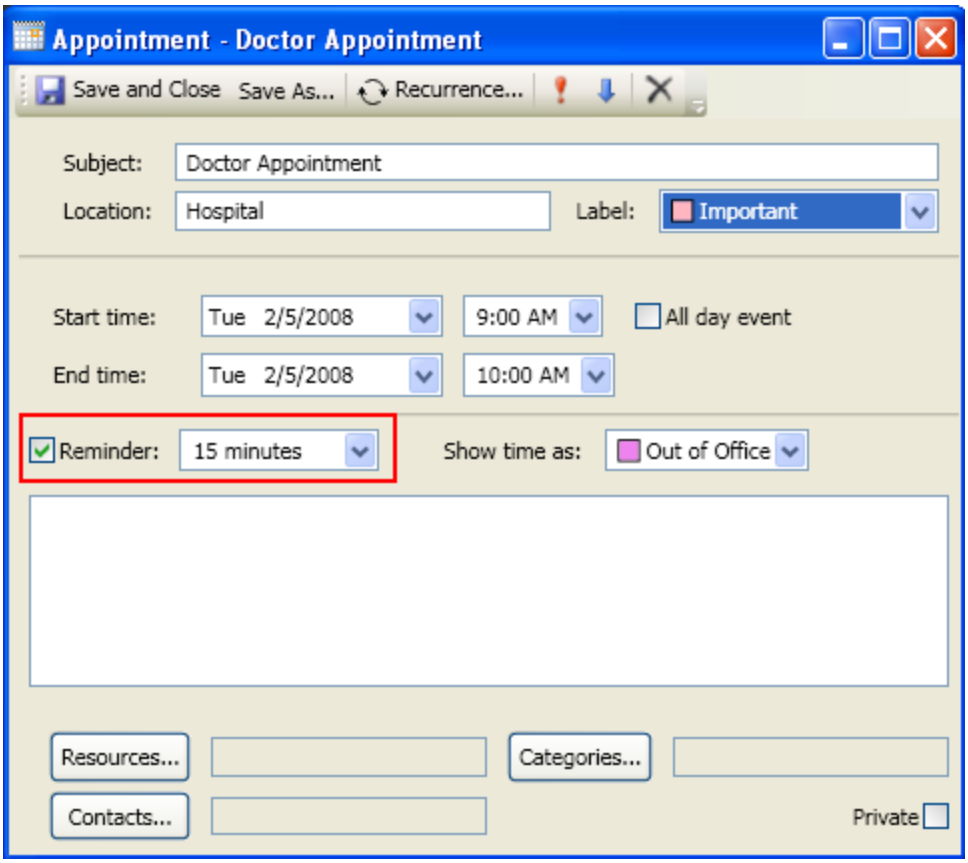


Reminders

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 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:

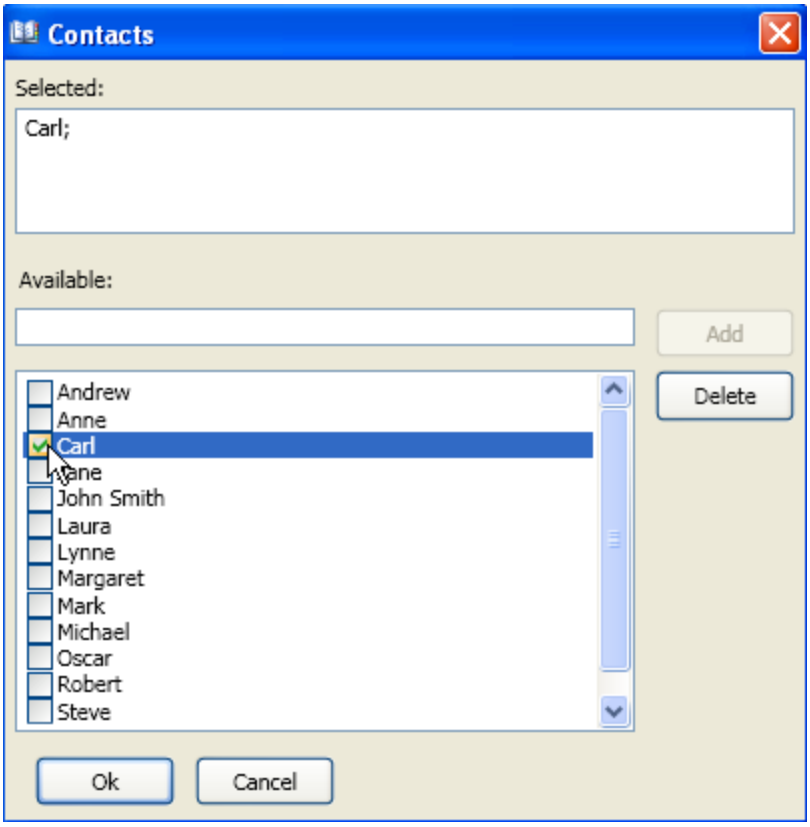
1. Add a new or open an existing appointment.
2. Click the **Contacts** button in the **Appointment** dialog box. The **Contacts** dialog box appears.
3. Enter a name in the **Available** text box and click **Add**.
4. Click **Ok** to close the **Contacts** dialog box.

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 (page 86).

To assign a contact to an appointment at run time:

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** 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

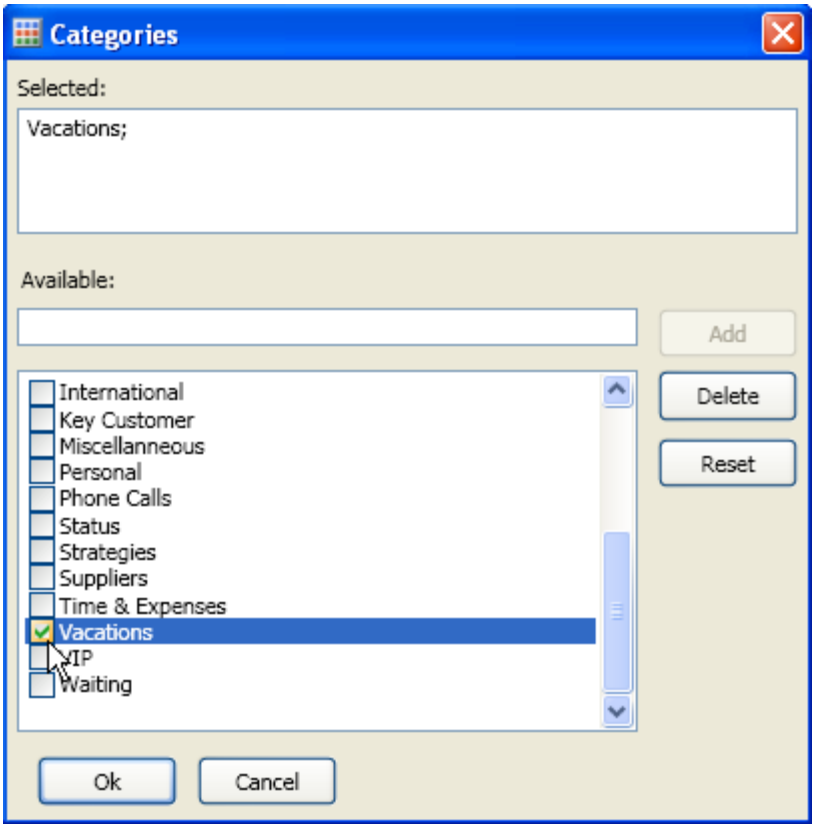
Adding Categories to the Categories List

Scheduler for WPF 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:

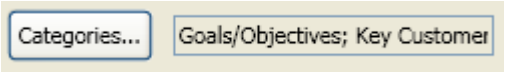


Assigning Categories to an Appointment

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 (page 88).

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

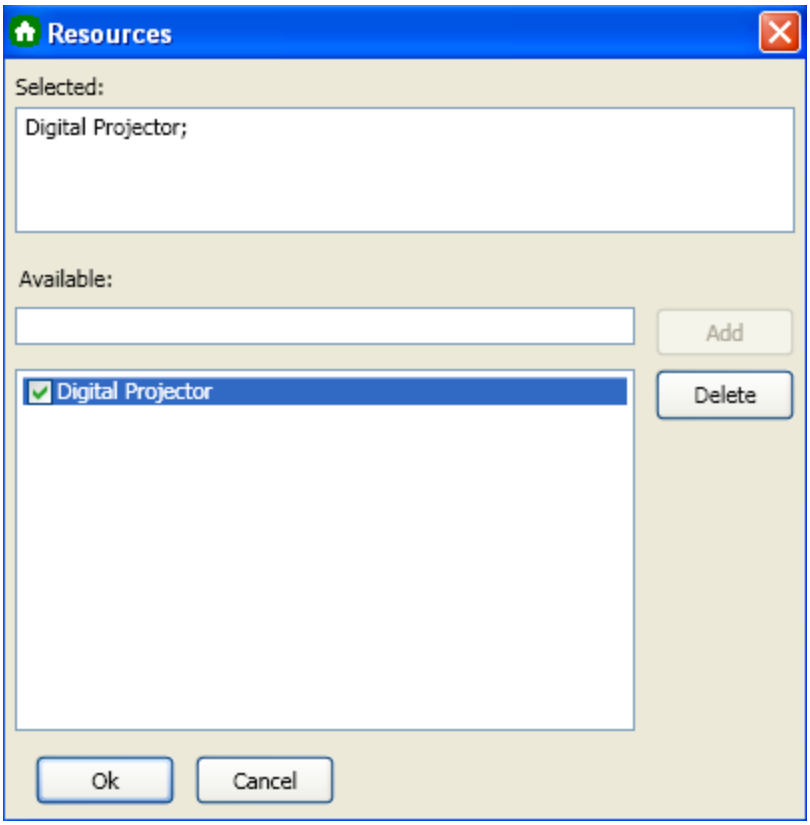
Scheduler for WPF supports resources created at run time through the **Resources** dialog box. Once added to the list, the resource can be assigned to an appointment.

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:

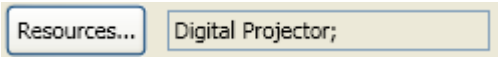


Assigning Resources to an Appointment

At run time, assigning a resource or multiple resources can be done through the **Resources** dialog box. By default, the **Resources** dialog box does not contain any resources. For information on how to add resources to the **Resources** dialog box, see Adding Resources to the Resources List (page 89).

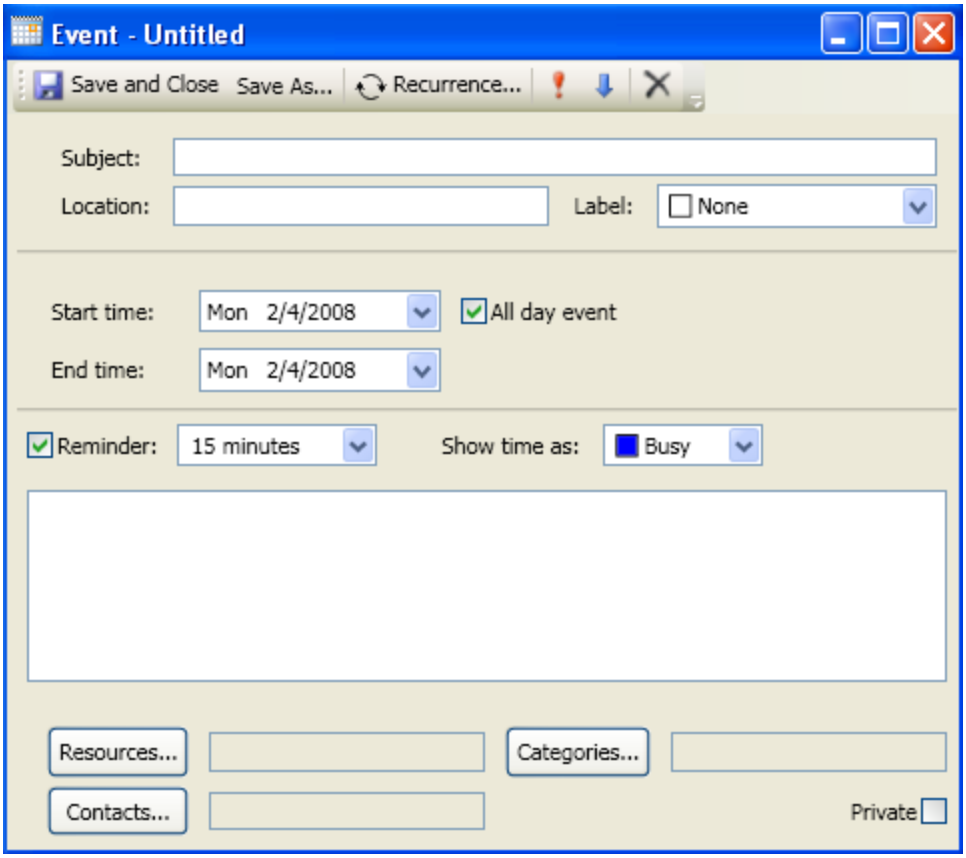
To assign a resource to an appointment at run time:

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



Working with Appointments at Run Time

Appointments can be created at run time by double-clicking the time for the appointment to begin, which will open the **Appointment** dialog box. For more information about appointments, see Appointments (page 81).



Pressing the following keys while in the **Appointment** dialog box will result in the following actions:

Key	Action
TAB or ENTER	Moves the cursor from one field to another according to the tab order.
ESC	Closes the **Appointment** dialog box without saving any changes.

Adding and Saving an Appointment

Appointments can be added to the schedule using the **Appointment** dialog box.

1. Double-click the time of the appointment, or select the appointment time and press the ENTER key. The **Appointment** dialog box appears.
2. Specify a **Subject**, **Location** and any additional information you would like to assign to the appointment.
3. Click the **Save** button to add the new appointment to the schedule.

Editing an Appointment

Appointments can be changed and updated through the **Appointment** dialog box.

1. Double-click an existing appointment to open the Appointment dialog box.
2. Edit any of the desired fields.
3. Click the **Save** button to update the appointment in the schedule.

Deleting an Appointment

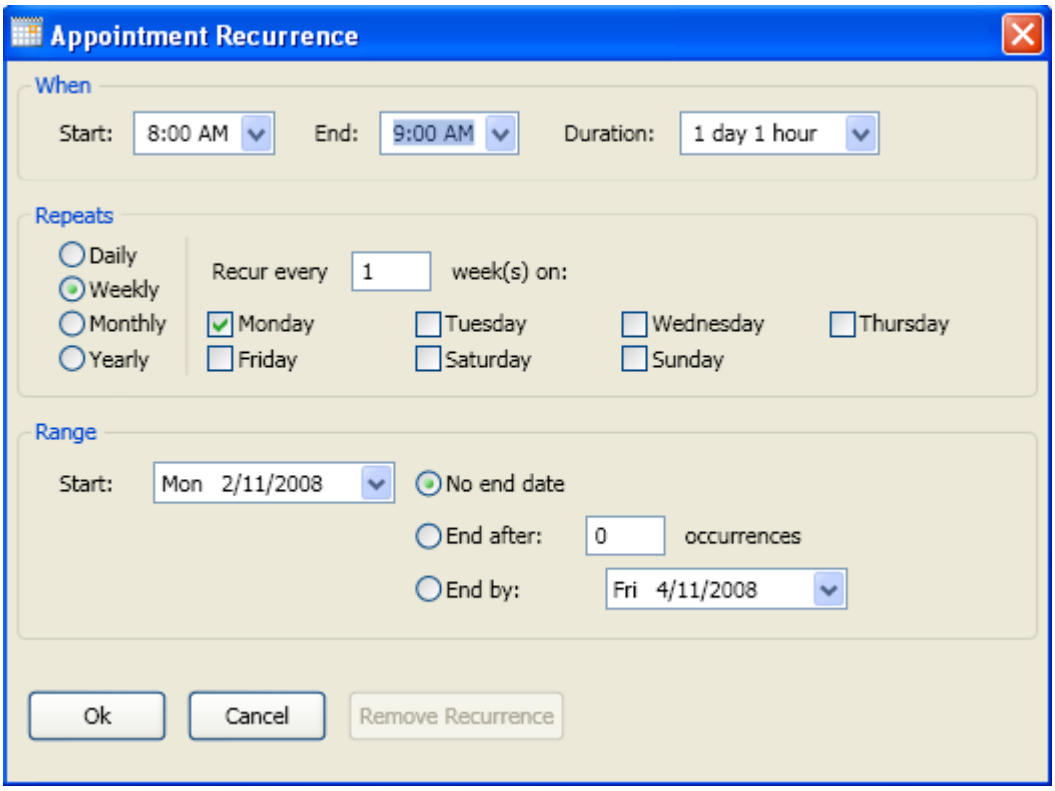
Selected appointments can be easily removed from your schedule through the **Edit Appointment** dialog box.

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.

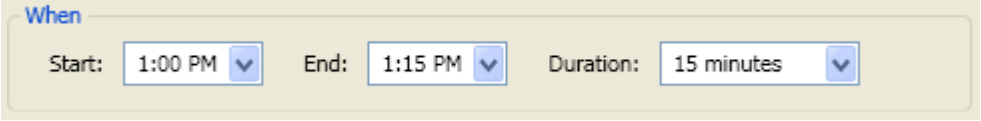
24. Double-click the time of an appointment to add a new one, or double-click an existing appointment. The **Appointment** dialog box appears.
25. Click the **Recurrence** button. The **Appointment Recurrence** dialog box appears.



26. Set the desired recurrence pattern:

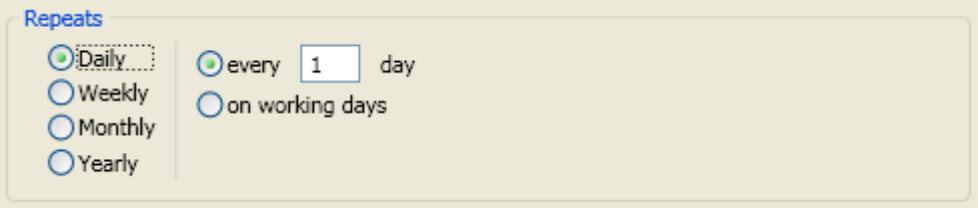
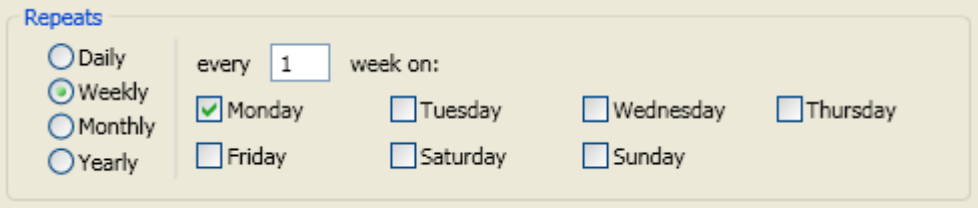
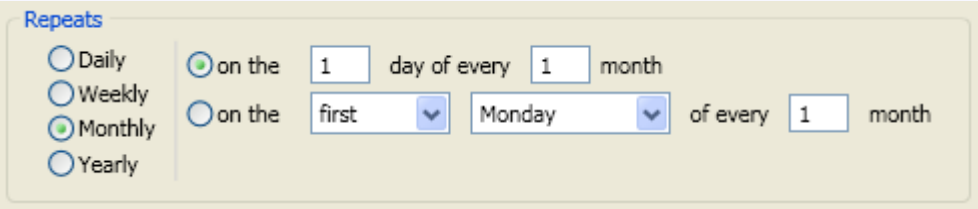
    When

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

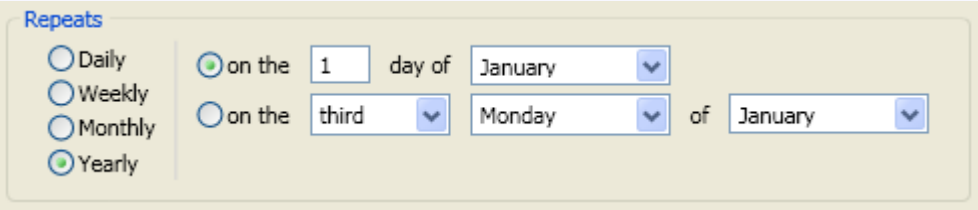


Repeats

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.

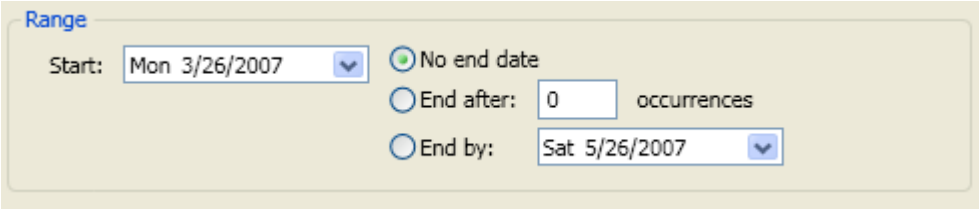


For example, setting the appointment to **on the 2 day of January** will make the appointment appear on January 2nd every year. This setting is ideal for birthdays and anniversaries.

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.

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

Scheduler for WPF Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with ComponentOne Studios.

Samples can be accessed from the ComponentOne Control Explorer. To view samples, on your desktop, click the **Start** button and then click **ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

The following pages within the ControlExplorer detail the **C1Scheduler** control:

C# Samples

Sample	Description
Themes and Views	Demonstrates how to use the themes, views, time scales, and work times for C1Scheduler.
Custom Styles	Shows how to create custom styles for the C1Scheduler control using custom defined styles instead of predefined C1Scheduler views.

Calendar for WPF Samples

Please be advised that this ComponentOne software tool is accompanied by various sample projects and/or demos, which may make use of other ComponentOne development tools included with ComponentOne Studios.

Samples can be accessed from the ComponentOne Control Explorer. To view samples, on your desktop, click the **Start** button and then click **ComponentOne | Studio for WPF | Samples | WPF ControlExplorer**.

The following pages within the ControlExplorer detail the **C1Calendar** control:

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.

Scheduler for WPF Task-Based Help

The task-based help assumes that you are familiar with programming in Visual Studio .NET and Microsoft Blend. By following the steps outlined in the Help, you will be able to create projects demonstrating a variety of Scheduler for WPF features and get a good sense of what Scheduler for WPF 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 assembly. For additional information on these topics, see Creating a .NET Project in Visual Studio (page 12), Creating a Microsoft Expression Blend Project (page 12), Adding the Scheduler for WPF Controls to a Blend Project (page 13), and Adding the Scheduler for WPF Components to a Visual Studio Project (page 13).

C1Scheduler Tasks

The following topics show how to perform specific C1Scheduler tasks.

Linking a Scheduler to a Calendar

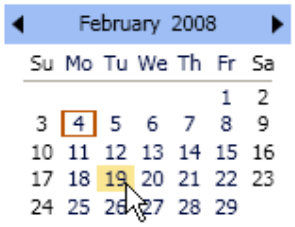
The following topic explains how to bind a schedule to a C1Calendar control in Microsoft Blend, Visual Studio, and using XAML.

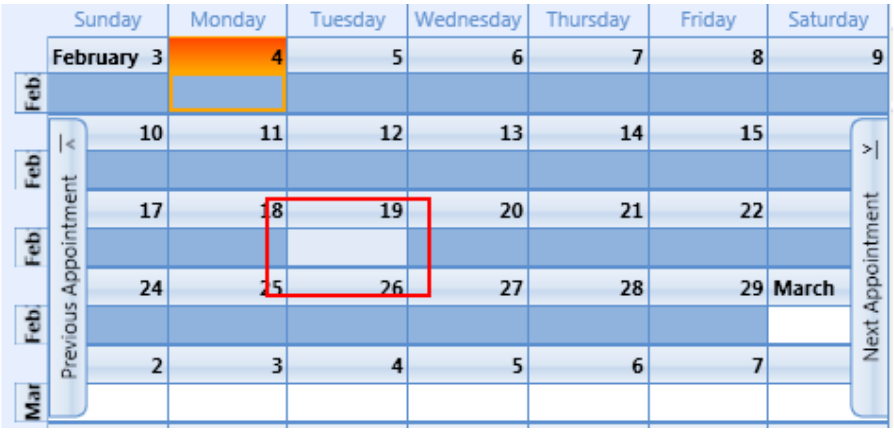
Using Microsoft Blend

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:
   ```
   <c1sched: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:
```
<c1sched: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:
```
<c1sched: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:
```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:c1sched="clr-namespace:C1.WPF.C1Schedule;assembly=C1.WPF.C1Schedule"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
```
4. Find and open **Default.xml** installed with ComponentOne Studio for WPF to C:\Program Files\ComponentOne\Studio for WPF\C1Schedule\XAML\themes\SchedulerThemes\Office2007.
5. Find the datatemplate with the C1Scheduler_TimeRuler_Template key. Click here to view the 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>
```

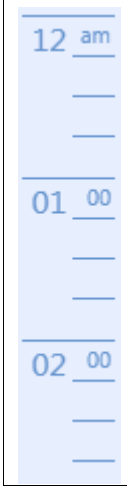
```
                              <ItemsPanelTemplate>
                                    <UniformGrid Rows="{Binding
ElementName=IntervalMarkersList, Path=Items.Count}" />
                              </ItemsPanelTemplate>
                        </ItemsControl.ItemsPanel>
                  </ItemsControl>
                  <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:
   ```
   <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
	

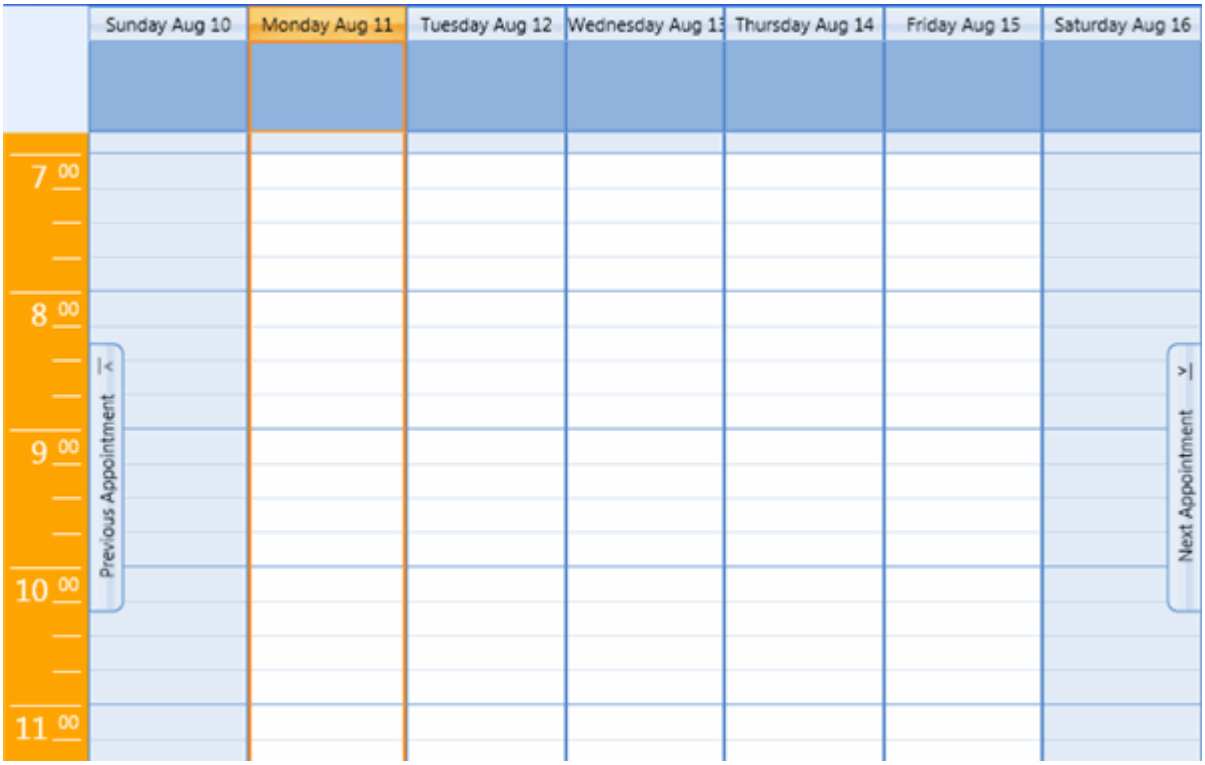
Changing the color of the time column

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:
```
<ResourceDictionary
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:c1sched="clr-namespace:C1.WPF.C1Schedule;assembly=C1.WPF.C1Schedule"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
```
4. Find and open **Default.xml** installed with ComponentOne Studio for WPF to C:\Program Files\ComponentOne\Studio for WPF\C1Schedule\XAML\themes\SchedulerThemes\Office2007.
5. Find the datatemplate with the C1Scheduler_TimeRuler_Template key. Click here to view the 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:
   ```
   <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:



Changing Navigation Pane Text

To change the navigation pane text, simply set the NextAppointmentText and 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 NextAppointmentText property.
3. Enter **Back** for the PreviousAppointmentText property.

Using Visual Studio

To change the navigation pane text in Visual Studio:

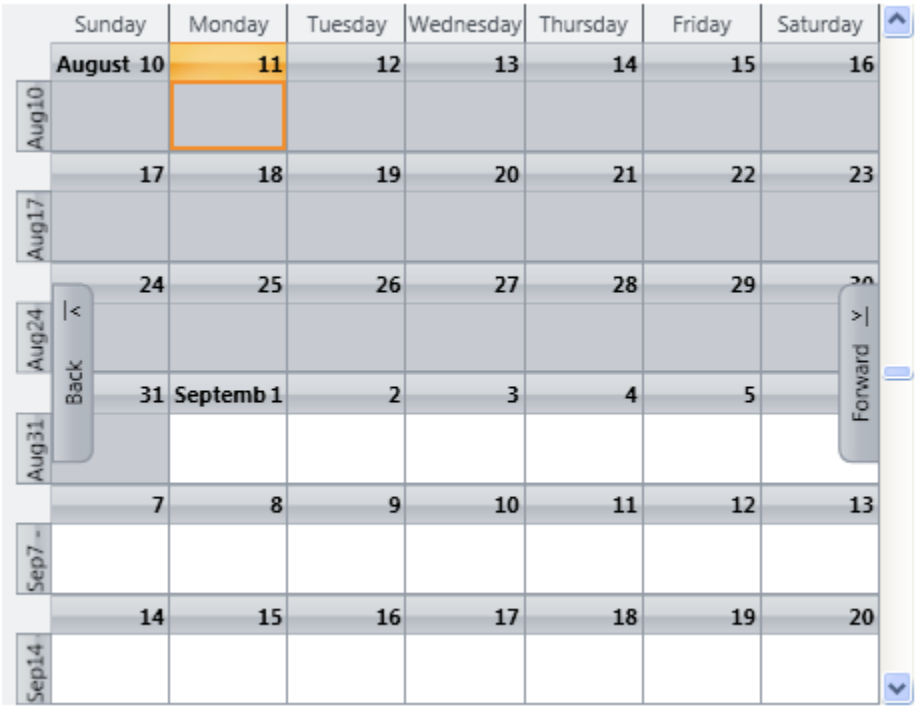
1. Add a C1Scheduler control to your window.
2. In the Properties window, set the NextAppointmentText property to **Forward**.
3. Set the PreviousAppointmentText property to **Back**.

Using XAML

The following XAML sets the NextAppointmentText and PreviousAppointmentText properties.

```
<c1sched:C1Scheduler Margin="15,15,0,12" Name="C1Scheduler2" Grid.ColumnSpan="2"
NextAppointmentText="Forward" PreviousAppointmentText="Back"
Theme="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler,
ResourceId=Office2007.Default}}"></c1sched:C1Scheduler>
```

The schedule will now look similar to the following image:



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 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 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**:

```
<c1sched:C1Scheduler x:Name="scheduler1" Theme="{DynamicResource
{ComponentResourceKey TypeInTargetAssembly=c1sched:C1Scheduler,
ResourceId=Office2007.Default}}" Style="{DynamicResource {ComponentResourceKey
TypeInTargetAssembly=c1sched:C1Scheduler, ResourceId=WorkingWeekStyle}}">
```

```
        <c1sched:C1Scheduler.CalendarHelper>
              <c1sched:CalendarHelper Culture="English " WeekStart="Sunday"
                     EndDayTime="18:20:00" StartDayTime="09:20:00"
                     WorkDays="Tuesday,Wednesday,Thursday,Friday,Saturday">
              </c1sched:CalendarHelper>
        </c1sched:C1Scheduler.CalendarHelper>
   </c1sched:C1Scheduler>
```

Grouping

You can easily start grouping by using the GroupBy property, which determines the type of grouping to display. The C1Scheduler control supports grouping by contacts, categories, resources, and by the 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:

```
<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" />
    <c1sched:C1Scheduler x:Name="sched1" Margin="195,0,12,62"/>

   </Grid>
</Window>
```

Double-click the **MainWindow** and add the following code to the **Window_Loaded** event:

- Visual Basic

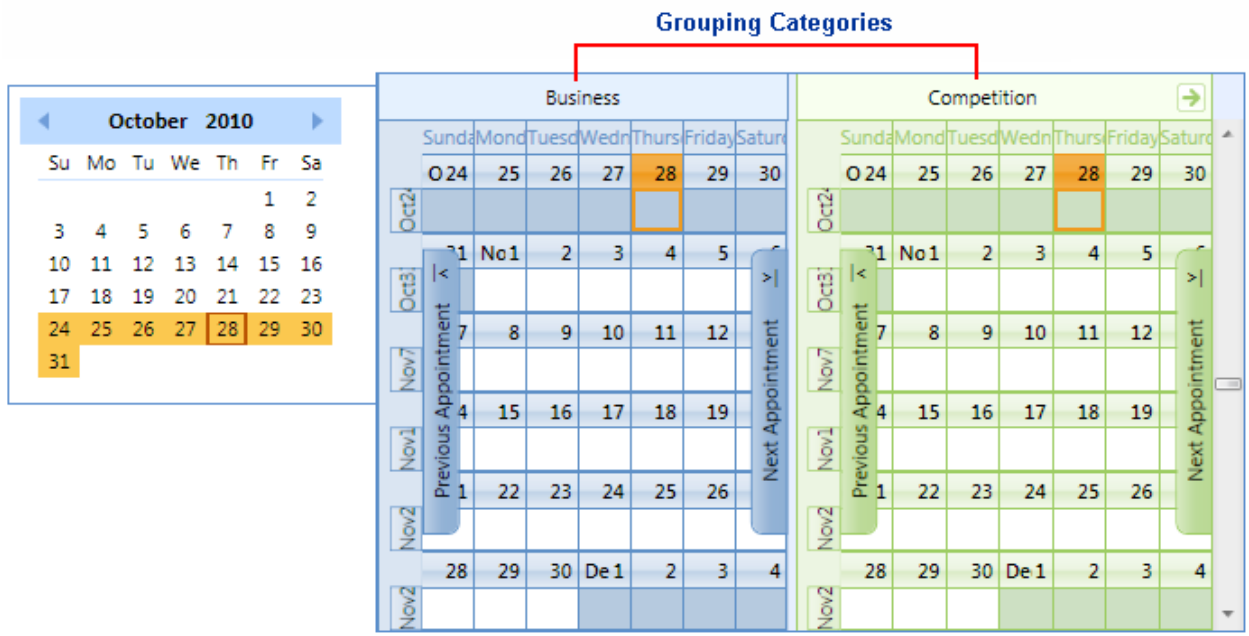
```
Private Sub Window_Loaded(sender As Object, e As RoutedEventArgs)
sched1.GroupBy = "Category"
End Sub
```

- C#

```
private void Window_Loaded(object sender, RoutedEventArgs e)
        {
            sched1.GroupBy = "Category";
```

```
}
```

When you run the project, the schedule will be grouped by categories, like in the following image:



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.

Adding Holidays to C1Calendar

You can add holidays to C1Calendar using the Holidays property from the CalendarHelper class.

Using Visual Studio

To add holidays to C1Calendar using the Holidays property from the CalendarHelper class in Visual Studio:

1. Add a C1Calendar control to your window and select it.
2. In the Properties window, expand the CalendarHelper item and click on the dropdown arrow next to Holidays. GenerateAdjacentMonthDays property.

Using XAML

The following XAML sets the GenerateAdjacentMonthDays property:

```
<c1sched: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.

Changing the Calendar Month or Year

You can easily change the selected month or year of a C1Calendar control using the SelectedDate property.

Using XAML

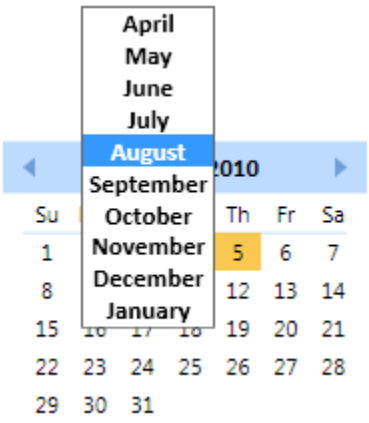
The following XAML sets the SelectedDate property:

```
<c1sched: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 MaxDate and 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 MaxDate property.
3. In the Properties window, enter '1/1/1700' next to the MinDate property.

Using XAML

The following XAML sets the MaxDate and MinDate properties:

```
<c1sched: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 Display

To show the previous and next month days of the month area, enable the 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 GenerateAdjacentMonthDays property.

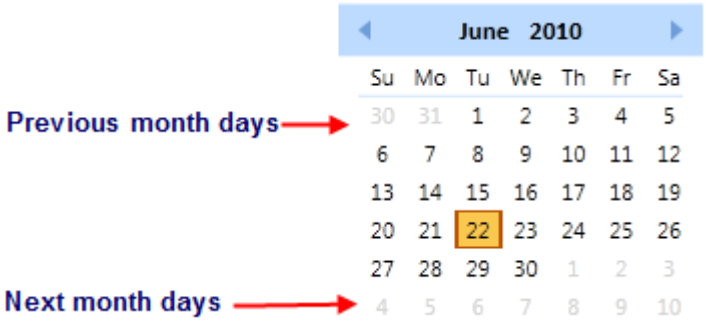
Using XAML

The following XAML sets the GenerateAdjacentMonthDays property:

```
<c1sched: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 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 MaxSelectionCount property.

Using XAML

The following XAML sets the MaxSelectionCount property to '31':

```
<c1sched:C1Calendar HorizontalAlignment="Left" Margin="10,10,0,0"
Name="c1Calendar1" VerticalAlignment="Top" MaxSelectionCount="31" />
```