



# TMS ASP.NET WEBPLANNER

## DEVELOPERS GUIDE

Apr 2010

Copyright © 2003 - 2010 by tmssoftware.com bvba  
Web: <http://www.tmssoftware.com>  
Email : [info@tmssoftware.com](mailto:info@tmssoftware.com)

**Supported .NET development environments** ..... 6

**Installation** ..... 7

**Installation of sample applications** ..... 9

**C# Samples for Visual Studio .NET 2003** ..... 9

**C# Samples for Visual Studio .NET 2005** ..... 9

**C# Samples for Visual Studio .NET 2008** ..... 9

**C# Samples for Visual Studio .NET 2010** ..... 9

**VB.NET Samples for Visual Studio .NET 2003** ..... 9

**VB.NET Samples for Visual Studio .NET 2005** ..... 9

**VB.NET Samples for Visual Studio .NET 2008** ..... 9

**VB.NET Samples for Visual Studio .NET 2010** ..... 9

**Introduction** ..... 11

**Overview** ..... 11

**What's New?** ..... 11

**Upgrade Notes** ..... 13

**System Requirements** ..... 14

**Upgrading** ..... 14

**Class Hierarchy** ..... 14

**Interfaces** ..... 15

**Globalization** ..... 15

**Installation** ..... 15

**Assemblies** ..... 16

        Microsoft Visual Studio.NET ..... 16

**Deployment** ..... 17

**Customization** ..... 18

        Debugging ..... 18

**Organization of this Manual** ..... 19

**License Agreement** ..... 19

**Frequently Asked Questions** ..... 20

**Customizing PlannerEvents** ..... 20

**Create custom hint windows** ..... 22

**Keep PlannerEvents from appearing in the WebPlanner** ..... 22

**Manage events without a DataStore** ..... 22

**Prevent Move or Resize** ..... 22

**Programmatically change an event's start and end time** ..... 22

**Security** ..... 22

**Getting Started..... 25**

**Geography and Terminology..... 25**

**Setting up the project ..... 28**

    Configuring the DataStore in ASP.NET 1.1 ..... 31

    Configuring the DataSource in ASP.NET2.0 or higher ..... 34

    Configuring the Controller ..... 35

**Customizing the WebPlanner ..... 37**

    Controlling the date ..... 37

    Specifying resources ..... 39

**WebPlanner..... 42**

**Create and Edit Events..... 42**

**Custom Editing..... 43**

    Detail Edit mode..... 43

    Custom Mode..... 47

**SideBar ..... 49**

**Header..... 51**

    Header Groups and SubGroups..... 52

**Bands and Zones..... 52**

**Positions ..... 54**

**Resources..... 55**

**Default Event..... 55**

**Items..... 55**

**Position Styles..... 56**

    Setting position colors..... 56

    Adjusting the active zone..... 56

    Establishing a no select zone ..... 57

**Layers ..... 57**

**Keyboard Support ..... 57**

**Client-side JavaScript ..... 58**

**Modes and Controllers..... 59**

**Day..... 59**

    MultiDay DayMapping..... 61

    MultiResource DayMapping..... 61

    MultiDayResource DayMapping..... 61

    MultiResourceDay DayMapping..... 61

**Day Period ..... 62**

**Half Day Period..... 62**

**Month ..... 63**

**MultiMonth..... 64**

**Timeline..... 65**

**Week..... 66**

**ActiveDay..... 67**

DisjunctDay .....	67
<b>MonthPlanner</b> .....	<b>69</b>
<b>Geography</b> .....	<b>69</b>
<b>Displaying Data</b> .....	<b>70</b>
<b>Create and Edit Events</b> .....	<b>70</b>
<b>Custom Editing</b> .....	<b>72</b>
Detail Edit Mode.....	72
Custom Mode.....	75
<b>Title</b> .....	<b>78</b>
Custom Glyphs.....	78
Title Style .....	79
<b>Day Header</b> .....	<b>79</b>
<b>Current Month Days</b> .....	<b>79</b>
<b>Other Month Days</b> .....	<b>81</b>
<b>Period Styles</b> .....	<b>81</b>
<b>Default Event</b> .....	<b>83</b>
<b>Items</b> .....	<b>83</b>
<b>Layers</b> .....	<b>83</b>
<b>Keyboard Support</b> .....	<b>84</b>
<b>Client-side JavaScript</b> .....	<b>84</b>
<b>Planner Events</b> .....	<b>85</b>
<b>Times</b> .....	<b>85</b>
<b>Captions and Notes</b> .....	<b>85</b>
<b>Behaviors</b> .....	<b>85</b>
<b>Colors</b> .....	<b>86</b>
<b>Flags</b> .....	<b>86</b>
<b>Hints</b> .....	<b>87</b>
Caption Hints .....	88
Custom Hints .....	88
<b>DataStores</b> .....	<b>91</b>
<b>Managing PlannerEvents by hand</b> .....	<b>91</b>
<b>Common DataStore functionality</b> .....	<b>92</b>
<b>SqlClientDataStore</b> .....	<b>93</b>
<b>OleDbDataStore</b> .....	<b>93</b>
<b>XmlDataStore</b> .....	<b>94</b>
Auto-creating a DataSet .....	94
Manually creating an Untyped DataSet.....	95
Data persistence .....	97
<b>Recurrency</b> .....	<b>100</b>

Database requirements ..... 100

Exceptions ..... 100

Editing recurrency ..... 101

*Using ClientEvents* ..... 103

*Built-in User rights management* ..... 107

    Rights on PlannerEvents ..... 107

    Rights on Planner positions ..... 108

*WaitList* ..... 109

    WaitList organization ..... 109

*Using Templates* ..... 111

*Index* ..... 115

## **Supported .NET development environments**

---

The TMS ASP.NET WebPlanner supports following development environments:

Microsoft™ Visual Studio .NET 2003  
Microsoft™ Visual Studio .NET 2005  
Microsoft™ Visual Studio .NET 2005 with Microsoft Ajax  
Microsoft™ Visual Studio .NET 2008 with Microsoft Ajax.  
Microsoft™ Visual Studio .NET 2010 with Microsoft Ajax.

The TMS ASP.NET WebPlanner supports following browsers:

Internet Explorer 6, 7, 8  
Firefox 3.0, 3.5  
Chrome 4.0

# Installation

---

Installation of components in the IDE

1)  
Execute SETUP.EXE  
This will install files under the folder:  
\\Program Files\\tmssoftware\\WebPlanner

2)  
**Visual Studio 2003:**

From the Visual Studio IDE menu, choose:  
Add & Remove toolbox items  
Choose Browse and pick WebPlanner.dll from the folder  
\\Program Files\\tmssoftware\\WebPlanner\\Bin\\VSNET2003

**Visual Studio 2005:**

From the Visual Studio IDE menu, choose:  
Tools,  
Customize Toolbox  
Switch to the .NET Framework Components tab  
Choose Browse and pick WebPlanner.dll from the folder \\Program Files\\tmssoftware\\WebPlanner\\Bin\\VSNET2005

**Visual Studio 2005 with Microsoft Ajax:**

From the Visual Studio IDE menu, choose:  
Tools,  
Customize Toolbox  
Switch to the .NET Framework Components tab  
Choose Browse and pick WebPlanner.dll from the folder  
\\Program Files\\tmssoftware\\WebPlanner\\Bin\\VSNET2005AJAX

(for the registered version with source code, either compile the solution for the IDE you're working with or download the precompiled assemblies from the registered users page on our website)

**Visual Studio 2008 with Microsoft Ajax:**

From the Visual Studio IDE menu, choose:  
Tools,  
Customize Toolbox  
Switch to the .NET Framework Components tab  
Choose Browse and pick WebPlanner.dll from the folder  
\\Program Files\\tmssoftware\\WebPlanner\\Bin\\VSNET2008

(for the registered version with source code, either compile the solution for the IDE you're working with or download the precompiled assemblies from the registered users page on our website)

**Visual Studio 2010 with Microsoft Ajax:**

From the Visual Studio IDE menu, choose:  
Tools,  
Customize Toolbox  
Switch to the .NET Framework Components tab  
Choose Browse and pick WebPlanner.dll from the folder  
\\Program Files\\tmssoftware\\WebPlanner\\Bin\\VSNET2010

(for the registered version with source code, either compile the solution for the IDE you're working with or download the precompiled assemblies from the registered users page on our website)

3)

WebPlanner & MonthPlanner should preferably be used with cached Javascript. JS client script files are in your webapplication path (when CachedScript is set to true), when starting a new web application that uses such components, copy the appropriate menu JS file to your webapplication path. The JS files are located in the folder: \Program Files\tmssoftware\WebPlanner\Scripts (registered version only)



## **Installation of sample applications**

---

Sample applications are provided for Visual Studio .NET for C# and VB.NET.

### [C# Samples for Visual Studio .NET 2003](#)

Copy the folder “\Program Files\tmssoftware\WebPlanner\DemoVSNET2003” into your IIS webserver path under virtual directory \DemosVSNet2003 (ie. normally c:\inetpub\wwwroot\DemosVSNet2003)  
From the IIS administration, create an application directory for this virtual directory. To run the DB demos which are based on Access files, it is required to set the folder c:\inetpub\wwwroot\DemosVSNet2003 as a shared folder. Index.aspx is the project startup file.

### [C# Samples for Visual Studio .NET 2005](#)

Choose : File, Open, “Web Site” from the Visual Studio IDE and browse to the folder “\Program Files\tmssoftware\WebPlanner\DemoVSNET2005” and start the application

### [C# Samples for Visual Studio .NET 2008](#)

Choose : File, Open, “Web Site” from the Visual Studio IDE and browse to the folder “\Program Files\tmssoftware\WebPlanner\DemoVSNET2008” and start the application

### [C# Samples for Visual Studio .NET 2010](#)

Choose : File, Open, “Web Site” from the Visual Studio IDE and browse to the folder “\Program Files\tmssoftware\WebPlanner\DemoVSNET2010” and start the application

### [VB.NET Samples for Visual Studio .NET 2003](#)

Copy the folder “\Program Files\tmssoftware\WebPlanner\DemosVBNET2003” into your IIS webserver path under virtual directory \DemosVBNet2003 (ie. normally c:\inetpub\wwwroot\DemosVBNet2003)  
From the IIS administration, create an application directory for this virtual directory. To run the DB demos which are based on Access files, it is required to set the folder c:\inetpub\wwwroot\DemosVBNet2003 as a shared folder. Index.aspx is the project startup file.

### [VB.NET Samples for Visual Studio .NET 2005](#)

Choose : File, Open, “Web Site” from the Visual Studio IDE and browse to the folder “\Program Files\tmssoftware\WebPlanner\DemoVBNET2005” and start the application

### [VB.NET Samples for Visual Studio .NET 2008](#)

Choose : File, Open, “Web Site” from the Visual Studio IDE and browse to the folder “\Program Files\tmssoftware\WebPlanner\DemoVBNET2008” and start the application

### [VB.NET Samples for Visual Studio .NET 2010](#)

Choose : File, Open, "Web Site" from the Visual Studio IDE and browse to the folder "\Program Files\tmssoftware\WebPlanner\DemoVBNET2010" and start the application

Important note:

When opening the demo applications in Visual Studio .NET or Delphi for .NET, make sure that:

- 1) In the References of the project, a reference is correctly pointing to webplanner.dll
- 2) The demo csproj.webinfo file is correctly pointing the path in your webserver where the project is installed.

# Introduction

---

## Overview

WebPlanner provides a first-class ASP.NET control for implementing a broad range of planning and scheduling solutions. Whether a project requires the creation of a single-user Personal Information Management (PIM) application or time planning for multiple resources such as hotel rooms, rental cars, and university courses, WebPlanner provides an open, highly-configurable interface that will suit the project's needs. The WebPlanner supports dynamic viewing of resources and scheduled events in a variety of modes. Events may be scheduled at any granularity of time within a day view. Broader pictures of allocated resources and appointments can be had via the day period, month, multi-month, timeline, and week views. All of these modes are available through a single WebPlanner control placed on a web page.

In ASP.NET 1.1 the WebPlanner is loosely coupled with its data and is more flexible than other databound web controls. By quickly connecting the WebPlanner to a SqlClientDataStore component, the WebPlanner will retrieve information from Microsoft SQL Server. If a project needs to use another database engine, the SqlClientDataStore can be quickly replaced with the OleDbDataStore component. For demos or single user applications, use the XmlDataStore component to have the WebPlanner manage data in a local XML file through the use of a DataSet. Because of the DataStore architecture, custom DataStores may be created and used with WebPlanner as necessary.

From ASP.NET 2.0, the WebPlanner utilizes the new DataSource components for seamless databinding. It is sufficient to create an OleDbDataSource or SqlClientDataSource with defined INSERT, UPDATE, DELETE commands and the WebPlanner can use it without code to read, create & update events from a database.

When used as is, the default WebPlanner has a nice visual appearance. But every visual aspect can be changed quickly and easily. One way is to use one of the predefined styles. Another way is to quickly tweak individual properties by hand. The sidebar displaying the time of day or other time unit can be shown on the top, left, right, or left and right of the WebPlanner. The sidebar may be configured to display different background colors depending upon whether the time slot is occupied by a scheduled item. The WebPlanner's displayed, active, and inactive periods may be configured such that each day has the same set of periods or certain days can have custom periods and coloring based upon criteria established by the application.

When it comes to the viewing and editing options available to the end user, WebPlanner has few competitors. For every scheduled event displayed to the user, an application using WebPlanner can control whether the event is visible and whether it can be resized, edited, or moved to a different time slot or day. The application may also choose whether the user edits events via an inplace editor or a popup editor. And it goes without saying that every aspect of an editor's visual appearance can be customized. One of WebPlanner's most distinctive features is that applications may assign scheduled events to different layers. The application may choose to display all events regardless of layer or display only those events assigned to one or more specific layers.

WebPlanner is a versatile product that will prove useful and valuable when used in your applications.

## What's New?

The following new features were added to WebPlanner v1.5:

- A new control named **MonthPlanner** provides a user interface similar to a wall calendar. It provides the same type of design time interface and drag and drop capabilities as the WebPlanner.
- Improved support for creating and editing PlannerEvents using a custom web form. By setting the **WebPlanner.EditType** property to EditTypes.DetailEdit and specifying information about the web form in the **WebPlanner.DetailEdit** property, WebPlanner will automatically display the form when the user is creating or editing an event.
- The WebPlanner and MonthPlanner client-side JavaScript may now be cached on the client.

- Rendering of the WebPlanner has been optimized to reduce the amount of HTML delivered to the client. This results in faster loading of the page by the client browser. The improvement is most noticeable for situations where large numbers of resources and events are being used.
- The WebPlanner may be restricted to a fixed height using the **WebPlanner.AutoSize** property. If the WebPlanner's body extends beyond the specified height, it will automatically display a scroll bar to the right of the body.
- The WebPlanner has a new **AutoEditOnCreate** event. When set to true, an event is automatically placed in edit mode after it is created.

#### The following new features were added to WebPlanner v2.0:

- New: Visual Studio 2005 support
- New: Drag indicator during moving of events in the WebPlanner
- New: Waitlist component from where unbound events can dragged into a WebPlanner or MonthPlanner
- New: Header Groups & SubGroups for better grouped indication of WebPlanner positions
- New: WebPlanner Recurrency support. Supports the vCalendar recurrency formulas to automatically handle recurrent events. Support automatic exception creation.
- New: ActiveDayController to see active days only, ie. most commonly weekdays only and skip weekend days.
- New: DisjunctDayController to see disjunct set of days allowing views such as every monday of the month etc..
- New: DetailEditEmbedded editing & view style allows inplace embedded detail forms for editing events.
- New: Ctrl-drag to copy events in WebPlanner
- New: Completion display to indicate completion per resource or day as progressbar in the WebPlanner
- New: SetCellColor delegate to dynamically customize WebPlanner cell background colors
- New: Custom glyph in event caption for custom client & server actions.
- New: client-side event right-click handling.
- New: Download all events as vCalendar file.
- New: Sidebar top & bottom display capability
- New: Header top & bottom display capability
- New: Custom delegate for custom JS callbacks
- New: Support for Apple browser Camino v0.9 or higher
- New: fine control for edit target & view target display to control view target on event click and edit target on event edit glyph click
- New: MonthPlanner ShowTitle property to enable or disable month title bar.

#### The following new features were added to WebPlanner v2.5:

- New: support for Microsoft Ajax in Visual Studio 2005

#### The following new features were added to WebPlanner v2.6:

- New: support for PlannerEvent owner ID and user ID for automatic rights management
- Methods CreateImageLinkRef, CreateImageHyperlink in PlannerEvent class
- Client side events SideBarClick, HeaderClick added
- Server side event SideBarClick added
- OverlapSpacerWidth property added
- PlannerEvent.Caption.Hint property added

- CaptionHintStyle Hint added
- EventKey parameter added for client event parameters

**The following new features were added to WebPlanner v2.7:**

- New: template support for view & edit of PlannerEvents
- New : events EventCopying, EventCopied added
- Improved : RefreshParentWindow method has been changed to do only WebPlanner control update in Ajax mode
- Improved : increased performance in Javascript rendering and reduced bandwidth usage

**The following new features were added to WebPlanner v2.7.5:**

- WaitList built-in databinding capability added
- Template support for MonthPlanner added
- EditType Template added (when EditType != Template only the ViewEventTemplate is shown)

**The following new features were added to WebPlanner v2.8.0:**

- New: Visual Studio 2008 support

**The following new features & improvements were added to WebPlanner v3.0:**

- New: ObjectDataSource support
- New: EventWidth property to set a fixed width for all overlapping PlannerEvents. The width of the Position containing the PlannerEvents is automatically adjusted
- New: AutoPositionWidth property. When set to true the width of the PlannerEvents and Positions is automatically adjusted according to the number of overlapping PlannerEvents relative to the total Planner width
- New: ShowNextMonth, ShowNextYear, ShowPreviousMonth, ShowPreviousYear properties added to enable/disable the browse buttons
- New: SetCellProp, SetHeaderCellProp and SetSideBarCellProp events added
- New: Several new ClientEvents added: CellDoubleClick, HeaderClick, HeaderRightClick, SideBarClick, SideBarRightClick
- Improved: More parameters have been added to the ClientEvents

**The following new features & improvements were added to WebPlanner v3.0.1.0:**

- New: Support for Visual Studio 2010, ASP.NET 4.0
- Improved : Internet Explorer 8.0 compatibility
- Fixed : Issue with SideBar during Ajax refresh in Chrome
- Fixed : Possible issue with programmatically inserting events

## Upgrade Notes

For ASP.NET 1.1, WebPlanner 2005 is fully compatible with v1.x. No special actions need to be taken to upgrade existing applications using the WebPlanner to the new version.

For upgrading existing WebPlanner v1.x applications in ASP.NET 1.1 to WebPlanner 2005 in ASP.NET 2.0, databinding will need to be thoroughly revised. As the DataSource is in ASP.NET 2.0 the preferred and recommended databinding technology, the DataStores are deprecated and can no longer be used. In most cases, porting the application to ASP.NET 2.0 will require the following steps:

- Remove all datastore components and related dataadapter, dataconnection, datacommand components
- Bind your database table(s) to the new DataSource components

- Attach the DataSource for the events table to the WebPlanner / MonthPlanner. Make sure to generate INSERT, UPDATE, DELETE commands for the datasource.
- Set the various fields like KeyField, StartTimeField, EndTimeField in the WebPlanner field properties.

## System Requirements

The following requirements are in effect for this version of WebPlanner:

1. A computer running Windows 2000, 2003, 2008, Windows XP, Windows Vista, Windows 7.
2. Visual Studio.NET 2003, Visual Studio .NET 2005 or Visual Studio .NET 2005 with Microsoft Ajax, Visual Studio 2008, Visual Studio 2010.
3. A version of Internet Information Services (IIS), compatible with the aforementioned Integrated Development Environments (IDE), must be installed and configured for ASP.NET 1.1, ASP.NET 2.0, ASP.NET 3.5, ASP.NET 4.0 development.
4. A hard drive with at least 20 MB of free space.

**Note:** If ZoneAlarm is running on the computer being used to test an application using WebPlanner, ZoneAlarm must be disabled. Otherwise, ZoneAlarm will prevent postback data from reaching IIS and cause errors in the WebPlanner server-side code.

## Upgrading

If an ASP.NET application uses a prior version of WebPlanner, it will contain a Register directive like the following:

```
<%@ Register TagPrefix="cswp" Namespace="TMS.WebPlanner" Assembly="WebPlanner,
Version=3.0.0.0, Culture=neutral, PublicKeyToken=4bc5eb9a373f42a1" %>
```

Note that this directive contains a hard-coded version of the WebPlanner assembly being used. This is because the WebPlanner assembly is strong-named. When upgrading to a new version of WebPlanner, the Register directive in each page using the new version must be updated to reference the new version. For example, when upgrading from WebPlanner 2.0 to WebPlanner 2.5 then the directive must be changed to the following:

```
<%@ Register TagPrefix="cswp" Namespace="TMS.WebPlanner" Assembly="WebPlanner,
Version=3.0.1.0, Culture=neutral, PublicKeyToken=4bc5eb9a373f42a1" %>
```

## Class Hierarchy

The WebPlanner class hierarchy is as follows. The namespace containing the class is suffixed to the class in parentheses. If the class is not followed by a namespace then it is contained within the same namespace as its parent class.

System.Object

**PlannerEvent** (TMS.WebPlanner)

    Control (System.Web.UI)

        WebControl (System.Web.UI)

**Controller** (TMS.WebPlanner)  
**DayController**  
**DayPeriodController**  
**HalfDayPeriodController**  
**MonthController**  
**MultiMonthController**  
**TimelineController**  
**WeekController**  
**ActiveDayController**  
**DisjunctDayController**  
**MonthPlanner** (TMS.WebPlanner)  
**WebPlanner** (TMS.WebPlanner)

MarshalByRefObject (System)  
 Component (System.ComponentModel)  
**BaseComponent** (TMS.WebPlanner)  
**DataStore**  
**BdpDataStore** (TMS.WebPlanner.BDS)  
**OleDbDataDataStore** (.NET 1.1 only)  
**SqlClientDataStore** (.NET 1.1 only)  
**XmlDataStore** (.NET 1.1 only)

## Interfaces

WebPlanner defines the following interfaces that may be of use to developers wanting to create custom components and controls for use with this product:

- **IController** - This interface defines the functionality required for components driving the mode and layout of the WebPlanner. For example, the DayController is an IController that places a WebPlanner in Day mode and specifies the current date, start time, and end time to be used.
- **IDataStore** - This interface defines the functionality required for components that retrieve data for use by the WebPlanner. For example, the SqlClientDataStore is an IDataStore that retrieves data from Microsoft SQL Server. (Not available in ASP.NET 2.0 version)
- **IPlanner** - Defines the functionality required to implement a control such as the WebPlanner or MonthPlanner. The PlannerEvent class will interface only with an object implementing the IPlanner interface.

## Globalization

The WebPlanner and MonthPlanner controls are culture-aware. When they need to produce day names, AM/PM designators, or formatted dates and times, they use the DateTime format information for the thread's current culture. By default, both WebPlanner and MonthPlanner change the current thread's culture to match the culture of the client web browser. This is done by looking at the first value available via the **HttpRequest.UserLanguages** property. To disable the automatic culture detection, set the WebPlanner's **AutoDetectCulture** property to the value **false**.

All of the messages displayed by the WebPlanner are customizable. The default messages are in English. However, they may be changed to a message in the appropriate language.

## Installation

The WebPlanner installation creates the following subfolders within the root installation folder:

1. **Bin** - Contains the official assemblies released for this version of WebPlanner.
2. **Doc** – This directory contains the PDF documentation provided with WebPlanner.

3. Examples – With WebPlanner 2005 the Examples have been removed from the installer. You can now download the most current example applications for the desired IDE and language from the TMS website at <http://www.tmssoftware.com/go/?webplanner>. Each example application comes with detailed instructions on installation and setup.
4. Source – The source code for the WebPlanner assemblies is contained within this subdirectory and its subdirectories. The source code doesn't ship with the evaluation and standard versions.

## Assemblies

The following assemblies are installed into the Global Assembly Cache (GAC):

- WebPlanner - Contains the run time support for the WebPlanner, MonthPlanner, controllers, and datastores. It contains late bound references to the WebPlanner.Design assembly. WebPlanner.dll is the ONLY WebPlanner dll that you are allowed to deploy to your run time environment.
- WebPlanner.Design - Contains the designers used by WebPlanner in Visual Studio. The design assembly is not deployable.
- AxShDocVw and ShDocVw - These assemblies provide an interface to the ActiveX WebBrowser control. They are referenced by the WebPlanner.Design assembly, generated using the *aximp* utility, and strong-named using the TMS key pair. They are for design time use only and are not deployable.

### Microsoft Visual Studio.NET

The installation automatically adds WebPlanner controls and components to the Visual Studio.NET 2003 Toolbox. To manually register WebPlanner with Visual Studio.NET in the event that the installer doesn't properly install everything, please do the following:

#### Visual Studio 2003

1. Start Visual Studio.NET.
2. Open the VS.NET Toolbox.
3. Right click on the Toolbox and choose the Add Tab menu item from the popup menu.
4. Enter the name "WebPlanner" into the new tab and press the Enter key.
5. Click on the new tab so that it displays its contents. At this time it should be devoid of components and controls.
6. Right click on the tab's client area and choose the Add/Remove Items menu item from the popup menu. The Customize Toolbox window appears.
7. In the Customize Toolbox window, click the Browse button. An Open dialog appears.
8. Navigate and select the file <installation root>\Bin\WebPlanner.dll. After clicking the OK button, the controls and components are added to the Toolbox.
9. Click the OK button to exit the Customize Toolbox window.

#### Visual Studio 2005

1. Start Visual Studio.NET 2005.
2. Create a new WebSite project and display the page designer.
3. If the installer didn't create a WebPlanner 2005 tab in your toolbox then create one by right clicking and selecting "Add Tab".
4. Right click on the tab name and select "Choose Items..."



5. When the "Choose Toolbox Items" dialog is displayed, Click the "Namespace" column header to sort by namespace and scroll to the TMS.WebPlanner namespace.
6. Place a checkmark next to each item in the TMS.WebPlanner namespace, and click [OK].
7. The WebPlanner tab will be filled with the controls.

#### Visual Studio 2008

1. Start Visual Studio.NET 2008.
2. Create a new WebSite project and display the page designer.
3. If the installer didn't create a WebPlanner 2008 tab in your toolbox then create one by right clicking and selecting "Add Tab".
4. Right click on the tab name and select "Choose Items..."
5. Navigate and select the file <installation root>\Bin\WebPlanner.dll. After clicking the OK button, the controls and components are added to the Toolbox.
6. When the "Choose Toolbox Items" dialog is displayed, Click the "Namespace" column header to sort by namespace and scroll to the TMS.WebPlanner namespace.
7. Place a checkmark next to each item in the TMS.WebPlanner namespace, and click [OK].
8. The WebPlanner tab will be filled with the controls.

#### Visual Studio 2010

1. Start Visual Studio.NET 2010.
2. Create a new WebSite project and display the page designer.
3. If the installer didn't create a WebPlanner 2010 tab in your toolbox then create one by right clicking and selecting "Add Tab".
4. Right click on the tab name and select "Choose Items..."
5. Navigate and select the file <installation root>\Bin\WebPlanner.dll. After clicking the OK button, the controls and components are added to the Toolbox.
6. When the "Choose Toolbox Items" dialog is displayed, Click the "Namespace" column header to sort by namespace and scroll to the TMS.WebPlanner namespace.
7. Place a checkmark next to each item in the TMS.WebPlanner namespace, and click [OK].
8. The WebPlanner tab will be filled with the controls.

## Deployment

Deployment of WebPlanner with your applications is simple and straightforward. In order for a production application to use WebPlanner, the WebPlanner assembly (i.e., WebPlanner.dll) must be deployed with the application. The assembly may be installed into the production machine's GAC or it may be installed as a private assembly in the application's bin directory.

## Customization

TMS ASP.NET WebPlanner Pro include the source code for the run time and design time portions of WebPlanner. Developers licensing the Pro edition have the freedom to customize the source code and to distribute an assembly containing the customized run time source code.

Some of the best improvements come from the people using the product. Developers also have the freedom to pass on customizations to TMS software via the [help@tmssoftware.com](mailto:help@tmssoftware.com) email. If the customizations have value for the general user population then TMS software will evaluate the changes and work to incorporate them into a future version of the product.

The remainder of this section discusses the strong naming of the WebPlanner assemblies and provides guidance for debugging your changes to WebPlanner.

### Debugging

Because the WebPlanner assemblies are installed into the GAC, customizing and debugging the source code requires certain procedures to be followed. The IDE, whether it is Visual Studio.NET or another supported development environment, must be aware of only one set of WebPlanner assemblies. If you attempt to use a debug version of the assembly while another version is installed in the GAC then you will encounter a variety of design time errors. For example, the WebPlanner control may fail to render on the design time form or certain type converters, collection editors, or property editors may work incorrectly or fail altogether.

Before customizing and/or debugging the WebPlanner source code, please do the following:

1. Remove the WebPlanner controls from the IDE's toolbox or palette.
2. Uninstall the WebPlanner assembly from the GAC. For example: `gacutil /u WebPlanner`.
3. If using Internet Information Services (IIS), restart the World Wide Web Service so that it will unload any instances of the WebPlanner assembly that may be in use.
4. Delete the temporary ASP.NET files in directory `C:\WINDOWS\Microsoft.NET\Framework\<version>\Temporary ASP.NET Files`. It's possible for information cached in these folders to confuse the IDE at design time.

At this point, you're ready to debug the WebPlanner assembly. Currently, WebPlanner includes project files for Visual Studio.NET (VS.NET). If you are using VS.NET, load the project file named *WebPlannerDebug.csproj*. This project file includes both the run time and design time code. In addition, its Debug configuration defines the conditional **DEBUGDESIGN**. When this conditional is active, the source code contains explicit references to the design time classes (e.g., `TypeConverter(typeof(PlannerEventConverter))`).

The assembly produced by this project can be used for both run time and design time customization and debugging. If you debug a canned WebPlanner example or one of your own projects that references the strong named WebPlanner assembly, you must remove the old WebPlanner reference and add a new one to the debug assembly produced by *WebPlannerDebug.csproj*.

If you changed only the run time source code, you can rebuild a strong named version of the WebPlanner assembly and use it with the existing *WebPlanner.Design* assembly. To do so, you must follow the basic steps for creating a public/private key pair and for associating the key pair with the assembly (see file *assemblyInfo.cs*). Because TMS software does not wish for developers to distribute assemblies that could be confused with those created by TMS software, developers may rebuild the WebPlanner assembly but must use their own public/private key pair.

Once the strong named WebPlanner assembly is compiled, you must install it into the GAC. Any projects or applications that were changed to reference the debug version of the assembly must now be switched back to the strong named assembly.

If you changed the design time source code, you must also build a strong named version of the *WebPlanner.Design* assembly. Again, this requires you to use your own private/public key pair. Note that the *WebPlanner.Design* assembly's version is fixed at "3.0.0.0" (or whatever is the version number of the release you're using). If you change

the version number, you must update the version number stored in constant **ProductInfo.StrongName**. This constant is located in the run time source code within file *ProductInfo.cs*. If this change is not made, the IDE will not be able to find the design time classes referenced by the run time source code.

For your design time changes to take effect, you must do the following:

1. Uninstall the WebPlanner assembly from the GAC.
2. Copy the customized WebPlanner assembly to the appropriate location. If you are going to leave it in the directory in which it was produced by the compiler, skip this step.
3. Install the custom WebPlanner assembly into the GAC.

The process of uninstalling and installing is important. If you do not uninstall, a cached version (i.e., the old version) of the assembly will be used and it will look like your changes have not taken effect.

## Organization of this Manual

The organization of this manual is as follows:

- Chapter 1 introduces WebPlanner and discusses its installation.
- Chapter 2 contains answers to frequently asked questions.
- Chapter 3 provides a short tutorial on how to use the WebPlanner, a DayController, and an XmlDataStore.
- Chapter 4 provides an in-depth treatment of the WebPlanner's user interface and configuration features.
- Chapter 5 discusses how to configure the WebPlanner to display information in various modes.
- Chapter 6 talks about the MonthPlanner control and how it may be configured.
- Chapter 7 presents information about the behaviors and color schemes for the events rendered in the WebPlanner and MonthPlanner.
- Chapter 8 describes how to integrate the WebPlanner with databases such as Microsoft SQL Server or Microsoft Access through the use of DataStores.
- Chapter 9 describes the WebPlanner built-in recurrency handling
- Chapter 10 shows how to use client events
- Chapter 11 explains user rights management with WebPlanner
- Chapter 12 talks about the WaitList component

## License Agreement

The license agreement for WebPlanner is included as a separate PDF document in the WebPlanner installation directory. To document is named *WebPlanner License.pdf*. It may be reached via the Start menu at Programs→TMS software→WebPlanner or via the Windows Explorer at *<installation root>\WebPlanner License.pdf*.

# Frequently Asked Questions

---

## Customizing PlannerEvents

### What is the best way to customize a PlannerEvent based upon its record in the database?

There are two parts to the answer. First, customization depends upon the application-specific information stored in the database. Typically, PlannerEvents are stored within a table of a database. The table must define certain fields required by the WebPlanner (e.g., Start time, end time). Any other fields present within the table wind up in the PlannerEvent's **OtherColumns** property. The OtherColumns property is of type HybridDictionary. The field names serve as keys for the HybridDictionary. The value associated with each key is the value found in the corresponding field. For example, if the table contains a field named "ReadOnly" then the application can access the value for the field as shown in the following code snippets:

#### C# Example

```
private void WebPlanner1_EventRetrieved(object sender,
    PlannerEventEventArgs e)
{
    bool isReadOnly = Convert.ToBoolean(e.Event.OtherColumns["ReadOnly"]);
}
```

#### VB.NET Example

```
Private Sub WebPlanner1_EventRetrieved(ByVal sender As Object, _
    ByVal e As TMS.WebPlanner.PlannerEventEventArgs) _
    Handles WebPlanner1.EventRetrieved

    Dim isReadOnly As Boolean
    isReadOnly = Convert.ToBoolean(e.Event.OtherColumns("ReadOnly"))

End Sub
```

There are two events in which it is best to act upon application-specific values associated with an event. One of those is the WebPlanner.EventRetrieved event as shown in the previous example. If a delegate or handler is provided for the event, the delegate or handler will be called once for each PlannerEvent retrieved from the database. The PlannerEvent may be accessed through the e.Event property. If the database contains a large number of events, using EventRetrieved may be inefficient. A better solution is to provide a delegate or handler for the WebPlanner.EventsRetrieved event. This event is raised after all PlannerEvents have been retrieved from the database. The events may be accessed through the e.Events property as shown in the following example:

#### C# EventsRetrieved

```
private void WebPlanner1_EventsRetrieved(object sender,
    TMS.WebPlanner.PlannerCollectionEventArgs e)
{
    // Color code the items based on the value of Layer.
    for (int i = 0; i < e.Events.Count; i++)
    {
        PlannerEvent item = (PlannerEvent)e.Events[i];
        object layer = item.OtherColumns["Layer"];
        item.Layer = Convert.ToInt32(layer);
        switch (item.Layer)
        {
```

```

case 0:
    item.Caption.BackColor = Color.WhiteSmoke;
    break;
case 1:
    item.Caption.BackColor = Color.Lime;
    break;
case 2:
    item.Caption.BackColor = Color.Blue;
    break;
case 4:
    item.Caption.BackColor = Color.Purple;
    break;
case 8:
    item.Caption.BackColor = Color.Firebrick;
    break;
case 16:
    item.Caption.BackColor = Color.Green;
    break;
default:
    item.Caption.BackColor = Color.Silver;
    break;
}
}
}

```

## VB.NET EventsRetrieved

```

Private Sub WebPlanner1_ItemsRetrieved(ByVal sender As Object, _
    ByVal e As TMS.WebPlanner.PlannerCollectionEventArgs) _
    Handles WebPlanner1.EventsRetrieved

    ' Color code the items based on the value of Layer.
    For I As Integer = 0 To e.Events.Count - 1
        Dim pItem As TMS.WebPlanner.PlannerEvent = e.Events(I)
        Dim layer As Int32 = pItem.OtherColumns("Layer")
        pItem.Layer = layer
        If pItem.Layer = 0 Then
            pItem.Caption.BackColor = Color.WhiteSmoke
        ElseIf pItem.Layer = 1 Then
            pItem.Caption.BackColor = Color.Lime
        ElseIf pItem.Layer = 2 Then
            pItem.Caption.BackColor = Color.Blue
        ElseIf pItem.Layer = 4 Then
            pItem.Caption.BackColor = Color.Purple
        ElseIf pItem.Layer = 8 Then
            pItem.Caption.BackColor = Color.Firebrick
        ElseIf pItem.Layer = 16 Then
            pItem.Caption.BackColor = Color.Green
        Else
            pItem.Caption.BackColor = Color.Silver
        End If

    Next I

End Sub

```

## Create custom hint windows

### Is it possible to create custom hint windows for an event?

Yes, you can define a custom hint for any PlannerEvent that needs it. The hint may be comprised of literal text or HTML markup. See the [Custom Hints](#) section for more information.

## Keep PlannerEvents from appearing in the WebPlanner

### How do I prevent certain PlannerEvents from being displayed in the WebPlanner?

The best way to do this is to provide a delegate or event handler for the WebPlanner.EventsRetrieved event. Within the handler, scan through the collection of events provided via the e.Events property. If an event should not be displayed in the WebPlanner, set its Visible property to the value **false**. For an example of providing a delegate for this event, see the previous section of this manual.

## Manage events without a DataStore

### How can my application manage events without using a DataStore?

The data stores provided with WebPlanner do not handle certain situations. For example, data stores do not handle situations where events must be retrieved and managed via stored procedures. Data stores also do not provide support for business layers. In those situations, the application can manage the PlannerEvents on its own by responding to events raised by the WebPlanner. For more details, see the section [Managing PlannerEvents by hand](#).

To see how an application could manage PlannerEvents without a DataStore, please have a look at the example installed with WebPlanner. The example is a single ASP.NET application comprised of several sub-applications. Before you run the example, you must set the start page to be Default.aspx. When you view Default.aspx in your browser, it will show you a list of the sub-applications. Choose the Meeting Rooms application. The Meeting Rooms application allows you to choose whether it uses a SqlDataStore, an OleDbDataStore, or a business layer.

## Prevent Move or Resize

### How do I prevent a PlannerEvent from being moved or resized?

The best way to do this is to provide a delegate or handler for the WebPlanner's **EventMoved** and **EventResized** events. In the handler for EventMoved, set the **e.MoveAllowed** property to the value false. In the handler for EventResized, set the **e.ResizeAllowed** property to the value false. This will force the PlannerEvent to remain at its current size and location.

## Programmatically change an event's start and end time

### When I change a PlannerEvent's StartTime and EndTime, the changes do not take affect. Why does the PlannerEvent appear in its previous position?

If an application changes the PlannerEvent's StartTime and/or EndTime properties, it must then call the PlannerEvent.UpdateTimes() method. This method calculates internal values that cause the PlannerEvent's location within the WebPlanner to be updated. To achieve the same effect in one method call instead, use the overloaded version: PlannerEvent.UpdateTimes(DateTime startTime, DateTime endTime).

## Security

### How do I prevent users from modifying events they do not own?

WebPlanner provides specific properties to control access to the WebPlanner and its events. The application is responsible for capturing the user login information and for correlating that information to the PlannerEvents (e.g., which events can user A modify and which events are read-only for that user?). The WebPlanner [online example](#) contains such a security system. The source code for the example is also available for [download](#).

If the WebPlanner should be read-only in its entirety (e.g., the user has not yet logged in but can see the WebPlanner) then the application can set the WebPlanner's **ReadOnly** property to the value true. The user will not be able to create events and they will not be able to edit or delete existing events.

Once the user has logged in, the application can mark certain PlannerEvents as read-only or mark entire [positions](#) as read-only. Marking individual events as read-only is useful when any given position of the WebPlanner contains events related to more than one user. To do this, the application must provide a delegate or event handler for the WebPlanner's **EventsRetrieved** event. Within that event, it compares the user login against the pertinent criteria on the PlannerEvent. If the user may not access an event then the application sets the PlannerEvent's ReadOnly property to the value true. Following is a code example illustrating this process:

### C# Example

```
// Holds the ID of the currently logged in user.
private int currentUserID = 0;

private void WebPlanner2_EventsRetrieved(object sender,
    PlannerCollectionEventArgs e)
{
    foreach (PlannerEvent plannerEvent in e.Events)
        // An event is read only if its userID field
        // does not match the current user.
        int userID = (Convert.ToInt32(plannerEvent.OtherColumns["UserID"]));
        plannerEvent.ReadOnly = (userID != currentUserID);
}
}
```

### VB.NET Example

```
' Holds the ID of the currently logged in user.
Dim CurrentUserID As Integer = 0

Private Sub WebPlanner1_EventsRetrieved(ByVal sender As Object, _
    ByVal e As TMS.WebPlanner.PlannerCollectionEventArgs) _
    Handles WebPlanner1.EventsRetrieved

    For I As Integer = 0 To e.Events.Count - 1
        Dim plannerEvent As TMS.WebPlanner.PlannerEvent = _
            e.Events(I)
        ' An event is read only if its userID field
        ' does not match the current user.
        Dim userID As Integer;
        userID = Convert.ToInt32(plannerEvent.OtherColumns("UserID"))
        plannerEvent.ReadOnly = userID <> CurrentUserID
    Next I

End Sub
```

If positions are tied to a user login (e.g., each position shown corresponds to a specific user) then the application can use [PositionStyles](#) to mark specific positions as read-only for the user. For example, if a scheduling application shows 3 positions with each position containing the appointments for 1 doctor then the application can mark 2 of the 3

positions as read-only, depending upon which doctor has logged in. The following code snippets illustrate this process:

### C# Example

```
// Holds the ID of the currently logged in user.
private int currentUserID = 0;

private void WebPlanner2_EventsRetrieved(object sender,
    TMS.WebPlanner.PlannerCollectionEventArgs e)
{
    // Assumptions:
    // Each position represents time scheduled for a specific user.
    // Each position has a corresponding resource representing the user.
    // Each position has a corresponding PositionStyle.
    for (int i = 0; i < WebPlanner2.Resources.Count; i++)
    {
        Resource resource = (Resource) WebPlanner2.Resources[i];
        PositionStyle positionStyle =
            (PositionStyle) WebPlanner2.PositionStyles[i];
        positionStyle.ReadOnly = (resource.ResourceID != currentUserID);
    }
}
```

### VB.NET Example

```
' Holds the ID of the currently logged in user.
Dim CurrentUserID As Integer = 0

Private Sub WebPlanner1_EventsRetrieved(ByVal sender As Object, _
    ByVal e As TMS.WebPlanner.PlannerCollectionEventArgs) _
    Handles WebPlanner1.EventsRetrieved
    ' Assumptions:
    ' Each position represents time scheduled for a specific user.
    ' Each position has a corresponding resource representing the user.
    ' Each position has a corresponding PositionStyle.
    For I As Integer = 0 To WebPlanner1.Resources.Count - 1
        Dim resource As TMS.WebPlanner.Resource
        resource = WebPlanner1.Resources(I)
        Dim positionStyle As TMS.WebPlanner.PositionStyle
        positionStyle = WebPlanner1.PositionStyles(I)
        positionStyle.ReadOnly = (resource.ResourceID <> CurrentUserID)
    Next I
End Sub
```



## Getting Started

---

This chapter walks through the creation of a simple application used to schedule the rides for a city's handicapped and disabled persons bus service. It shows how to connect and configure the required controls. While not representative of what would be done for a real life, multi-user application, it uses a local XML file to manage the data. Once the application is up and running, this chapter walks through customization of certain WebPlanner features. This chapter uses Visual Studio.NET 2003 to construct the project. The process is almost identical for those using Visual Studio 2005.

### Geography and Terminology

Much of this manual is filled with terminology and ideas specific to the WebPlanner control. The WebPlanner provides a visual interface for viewing and managing items that are scheduled for a certain time period. This section introduces you to the visual layout of the WebPlanner and the terms used to describe the different parts of the interface. The following screenshot shows some of the features typically seen in the WebPlanner. The subsequent text describes the numeric landmarks shown on the screenshot.

	Dr. Robinson	Dr. Atkinson	Dr. Johnson	Dr. Simpson	
		9:30 AM - 12:00 This is a header item.		This read-only header item has been created without a caption.	
7:00 AM					7:00 AM
7:30 AM					7:30 AM
8:00 AM					8:00 AM
8:30 AM					8:30 AM
9:00 AM		9:00 AM - 12:30 Notes may be changed via inplace editing.			9:00 AM
9:30 AM					9:30 AM
10:00 AM					10:00 AM
10:30 AM			10:30 AM - 12:00 PM This is a read-only item.		10:30 AM
11:00 AM					11:00 AM
11:30 AM					11:30 AM
12:00 PM					12:00 PM
12:30 PM					12:30 PM
1:00 PM				1:00 Overlapping allowed.	1:00 PM
1:30 PM					1:30 PM
2:00 PM				2:00 Overlapping allowed.	2:00 PM
2:30 PM	2:30 PM - 5:00 PM This item has a fixed position. It cannot be moved to another position.				2:30 PM
3:00 PM					3:00 PM
3:30 PM					3:30 PM
4:00 PM					4:00 PM
4:30 PM					4:30 PM
5:00 PM					5:00 PM
5:30 PM					5:30 PM
6:00 PM					6:00 PM
6:30 PM					6:30 PM
7:00 PM					7:00 PM
7:30 PM					7:30 PM
8:00 PM					8:00 PM
8:30 PM					8:30 PM
9:00 PM					9:00 PM
9:30 PM					9:30 PM
10:00 PM					10:00 PM
10:30 PM					10:30 PM

## 1: SideBar

The *sidebar* displays the time axis for the WebPlanner. The meaning and values of the time axis depend upon the WebPlanner's current *mode*. In the previous screenshot, the WebPlanner is in Day mode and the sidebar is divided into *time slots* representing the hours and minutes of the day. The sidebar may be positioned to the left, right, both left and right, or the top of the WebPlanner. The back color of the sidebar may differ based upon whether an item is occupying that time slot.

The following screenshot shows the same WebPlanner with the sidebar positioned on the top.

		7:00 AM	7:30 AM	8:00 AM	8:30 AM	9:00 AM	9:30 AM	10:00 AM	10:30 AM	11:00 AM	11:30 AM	12:00 PM	12:30 PM	1:00 PM
Dr. Robinson														
Dr. Atkinson	9:30 AM - 12:00 PM This is a header item.					9:00 AM - 12:30 PM Notes may be changed via inplace editing.								
Dr. Johnson								10:30 AM - 12:00 This is a read-only item.						
Dr. Simpson	This read-only header item has been created without a caption.													1:00 Ove a

## 2: Header

The header identifies the day and/or resource with which the scheduled events are associated. In certain modes, the header may contain *header events*. For example, an event in Day mode that is scheduled for the day, but not a specific period of the day, will be rendered in the header.

## 3: Position & Resource

Scheduled events are rendered within a *position*. When the sidebar is on the left and/or right of the WebPlanner, a position is rendered as a column. When the sidebar is on the top of the WebPlanner, the position is rendered as a row. A position may represent a day, a month, or a *resource*, depending upon the WebPlanner's *mode*. In the previous screenshots, the WebPlanner is in Day mode and its DayMapping is set to MultiResource. In this situation, each position represents a resource.

A *resource* represents the person or thing whose time is being scheduled. In this example, the time of several doctors is being scheduled. In other applications, the resource may be a meeting room, a car, a university classroom, a vacation home, or anything else that you can think of.

## 4: Cells

The main portion of the WebPlanner is divided into a grid of *cells*. Each cell corresponds to a time slot of the sidebar. Each cell is located within a *position*. A scheduled item may occupy one or more cells. To add a scheduled event, either click on a cell or select a contiguous range of cells. When the sidebar is positioned on the left and/or right of the WebPlanner, the range of cells must be vertical. When the sidebar is positioned on the top of the WebPlanner, the range of cells must be horizontal.

## 5: PlannerEvent

When a period of time is scheduled for a resource, it is represented by an instance of the *PlannerEvent* class. A *PlannerEvent* is rendered on the WebPlanner. A *PlannerEvent* has a caption and a body. The caption contains the subject of the *PlannerEvent* and the body contains its notes. The WebPlanner calculates the event's position, starting cell, and ending cell. The WebPlanner may be configured to allow a scheduled event to be changed via an in-place editor or a popup editor window. The application may mark a *PlannerEvent* as read only, as having a fixed start time, or a fixed position.

## 6: Overlap Column

When overlapping is enabled, each position is immediately followed by a thin *overlap column*. If a time slot is filled by one or more *PlannerEvents*, the user may create a new *PlannerEvent* by clicking on the overlap column cell adjacent to the cell in which the new *PlannerEvent* is to be added. Use *OverlapColor* property of the WebPlanner to control the overlap column's color.

## 7: Overlapping

When two or more *PlannerEvents* occupy the same or all of the same time period, they are said to be *overlapping*. Overlapping is enabled by default at the WebPlanner level. The application may disable overlapping for individual *PlannerEvents* as needed.

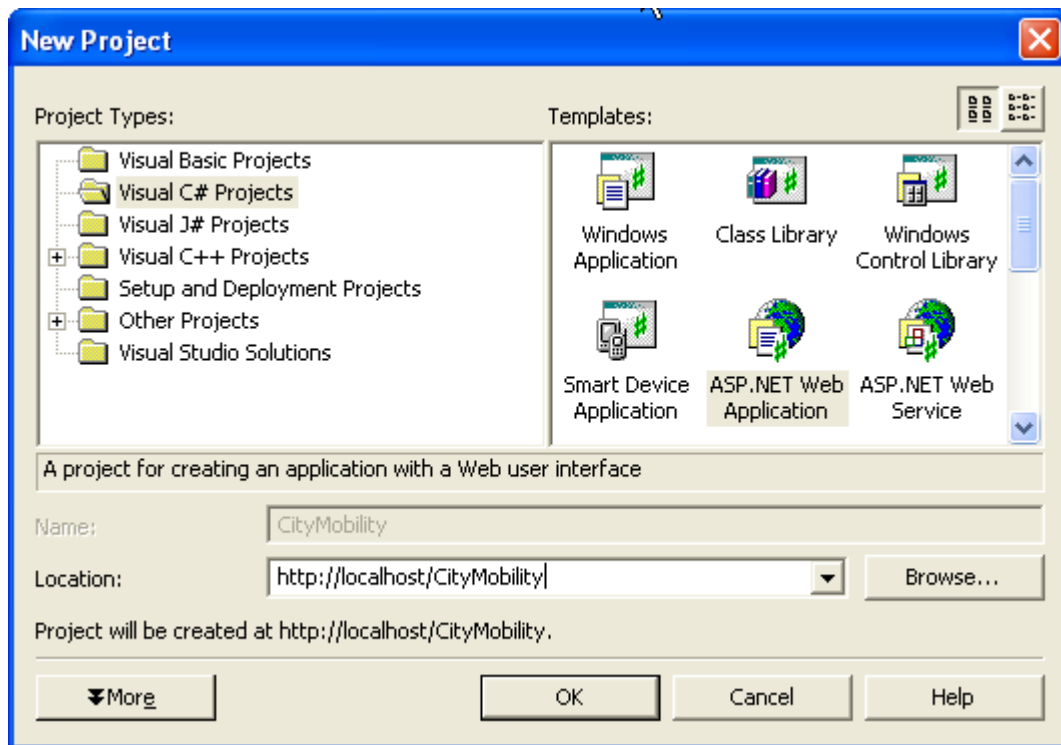
## 8: TrackBar

Each *PlannerEvent* is preceded and followed by a thin line called a *trackbar*. Use the trackbar to resize the event. To resize the event, move the mouse over the appropriate trackbar. When the mouse is drawn as a resize glyph, press and hold the left mouse button. Move the mouse in the appropriate direction and release the button when the correct time slot is reached. The application may disable resizing for specific *PlannerEvents* on an as-needed basis.

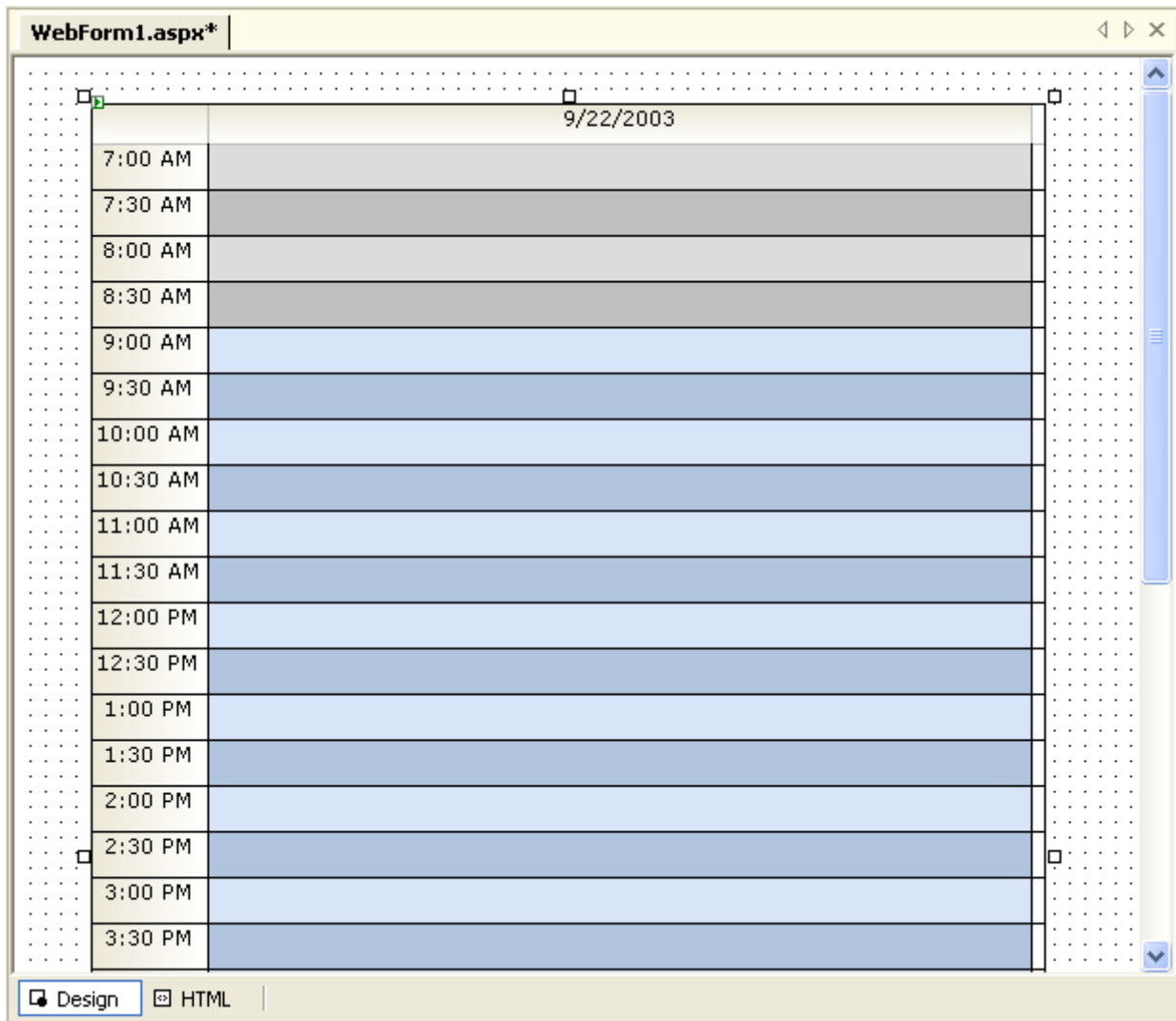
## Setting up the project

As mentioned in the Overview, the project will provide a fictitious government agency named CityMobility with the ability to schedule rides for people with disabilities. A person who qualifies for the program calls the CityMobility office and talks to a dispatcher to set up one or more rides. The dispatcher will enter the rides into the application via the WebPlanner. Each ride will be assigned to a specific driver.

After starting your development environment, create a new ASP.NET Web Application in the location <http://localhost/CityMobility>. The following screenshot shows the project creation in Visual Studio.NET.

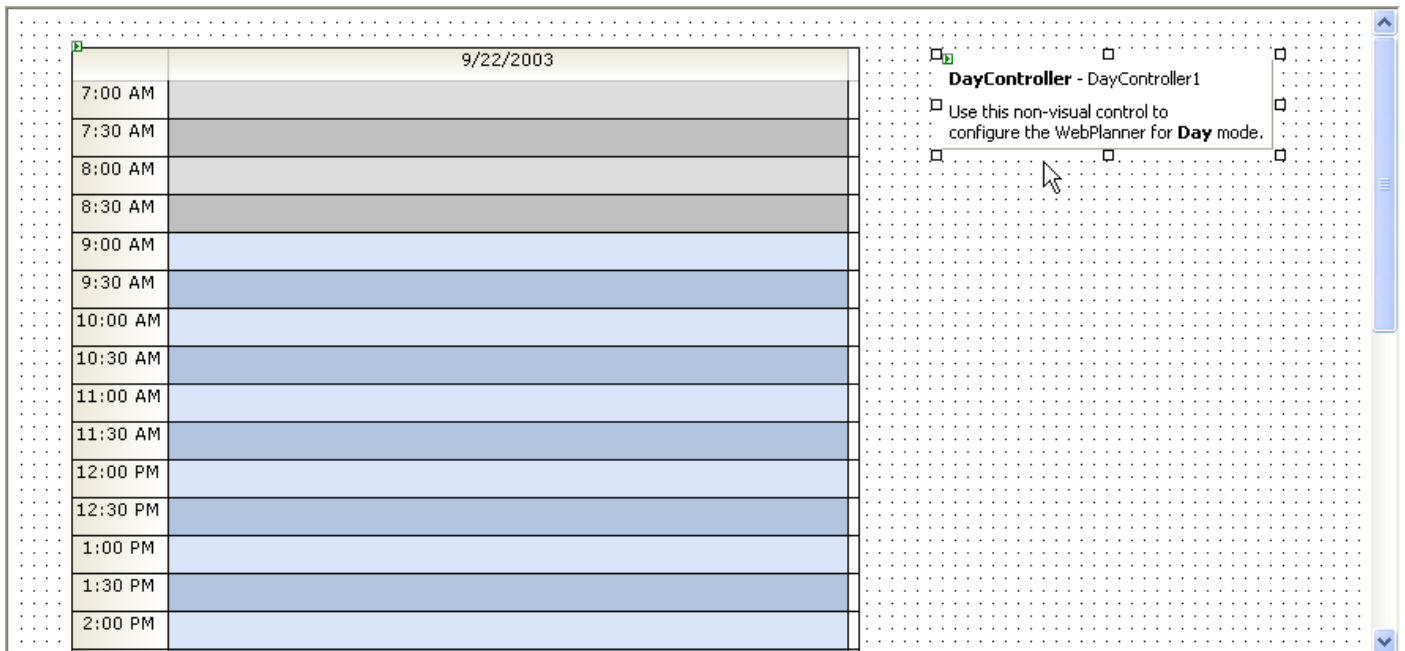


Go to the WebPlanner tab in the Toolbox and drop a WebPlanner control onto the web form. By default, the WebPlanner renders itself in Day mode with time slots displayed for 7:00 am through 10:30 pm. While events may be scheduled within any visible time slot, the WebPlanner displays the region from 9:00 am through 8:30 pm as an active zone. The active zone indicates where the user should schedule events. Later on in this chapter, we'll use a DayController to configure the active zone. The following screenshot shows our newly-placed planner and the time zones.

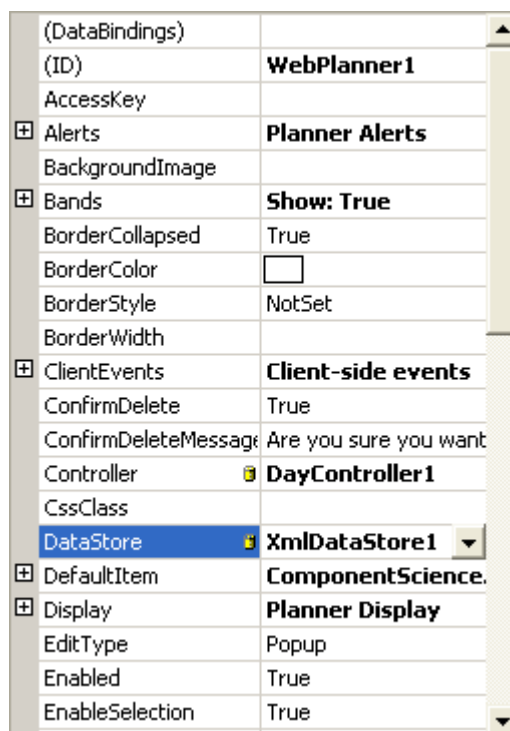


The WebPlanner needs data in order to be useful. While we could programmatically add and remove scheduled rides at run-time, it is far easier to allow the rides to be managed by a database. For the purposes of this project, we can get by with a single user database for demonstration purposes. When the project is fully implemented, we can switch to something like Microsoft SQL Server. To implement the single user database, drop an XmlDataStore component. In the Visual Studio.NET Toolbox, it will be the last component on the WebPlanner tab. We'll configure the XmlDataStore in the next section.

We also know that we will be scheduling rides on an hourly basis, so the WebPlanner will stay in its current Day mode at run-time. The DayController control provides us with a set of properties to easily configure the WebPlanner in Day mode. Drop a DayController onto the web form. The DayController renders itself on the web form at design-time, as shown in the following screenshot, but it will be invisible at run-time.



With the XmlDataSource and DayController in place, it is time to connect them to the WebPlanner. Select the WebPlanner on the form and go to the Properties page. For the WebPlanner's Controller property, select DayController1. For the WebPlanner's DataStore property, select the XmlDataSource1 component.



At this point, you have placed the necessary controls and components on the form. The next section walks through the configuration of the XmlDataSource, establishing a table and XML document in which the scheduled rides are stored.

Configuring the DataStore in ASP.NET 1.1

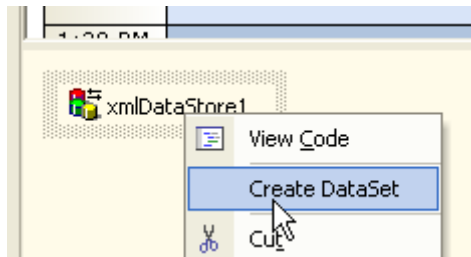
The XmlDataStore will use a DataSet to persist the bus rides, writing them to and reading them from an XML document. As mentioned previously, the XmlDataStore is primarily for giving demonstrations to customers and for creating single user applications. If a project must support multiple users, a more robust database engine is called for and it can be accessed via the SqlClientDataStore or OleDbClientStore components.

The WebPlanner has few requirements regarding how the scheduled items are stored in the database. The requirements are as follows:

1. The scheduled events must be stored in a single table.
2. The scheduled events must be uniquely identified via some type of key field.
3. Each scheduled event must have a start time and an end time.

Other than those requirements, the WebPlanner does not care what the table or its fields are named. As a matter of fact, you tell the WebPlanner those details via the XmlDataStore. The XmlDataStore can be used to persist scheduled events without writing any code. In the rest of this section, you will have the XmlDataStore create a DataSet and learn how the XmlDataStore persists the contents of the DataSet.

The XmlDataStore provides a "Create DataSet" verb that, at design time, programmatically creates a DataSet compatible with the datastore and connects the DataSet to the datastore. To create the DataSet, right click the XmlDataStore and select "Create DataSet" from the popup menu.



The XmlDataStore designer creates a new DataSet component having the name dataSet1. The DataSet contains a table named Items containing the following fields:

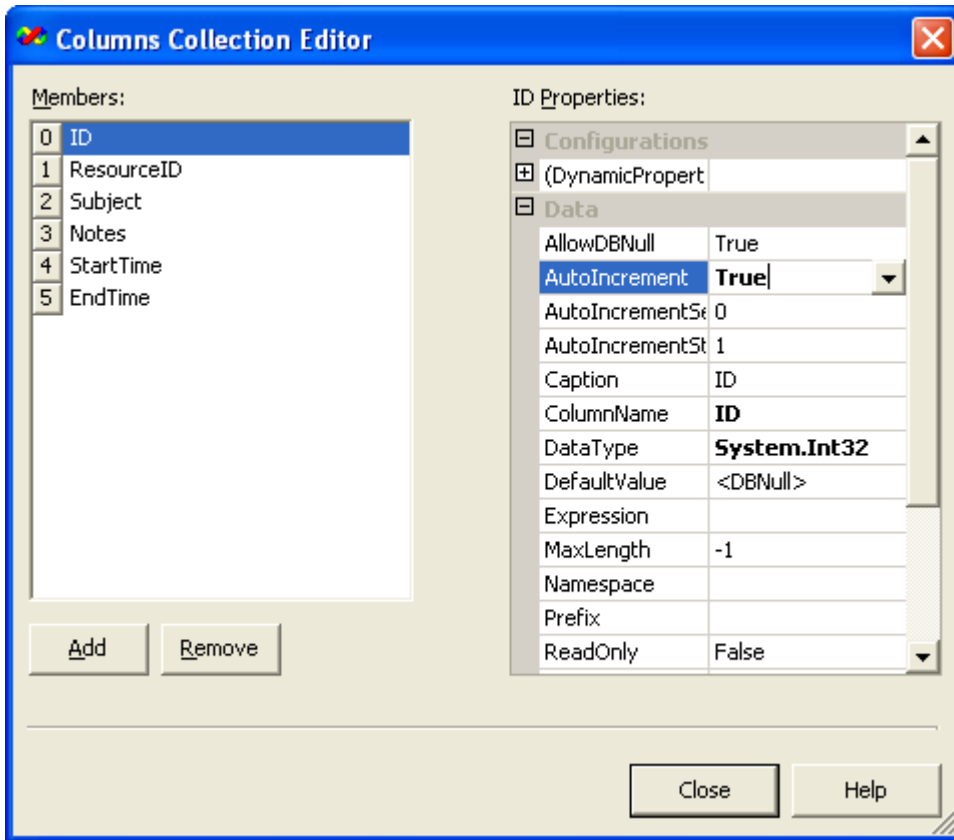
Column name	DataType	Intended Use
ID	System.Int32	This is the unique identifier for a scheduled ride. It is of type integer and the column's AutoNumber property is set to true.
Subject	System.String	The information displayed in the caption of the event when displayed on the WebPlanner.
ResourceID	System.Int32	The unique ID of the bus driver with which this ride has been scheduled.
Notes	System.String	Miscellaneous information associated with the scheduled ride. For example, this could contain the address at which the person is picked up and the address at which they are to be dropped off.
StartTime	System.DateTime	When the person is to be picked up.
EndTime	System.DateTime	The estimated drop off time.

To view the columns, do the following:

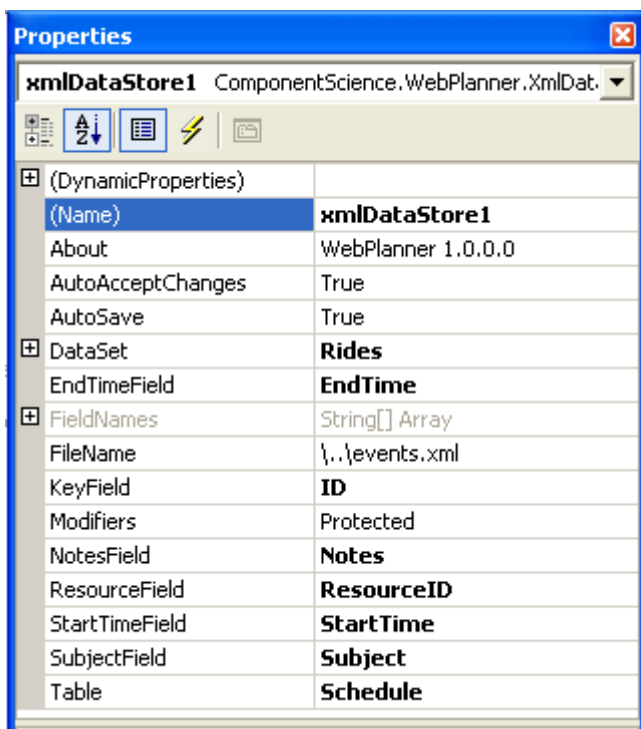
1. Select the DataSet and go to the Properties page.
2. In the Properties page, open the collection editor for the DataSet's Tables property.
3. In the Tables collection editor, open the collection editor for the Columns properties.



The Columns collection editor should appear as follows:



After the DataSet is created, the XmlDataStore designer also connects the DataSet to the XmlDataStore. In the IDE, select the XmlDataStore and go to the Properties page. The properties for the DataStore should appear as follows:



At this point in the project, the WebPlanner is able to read and write information via the XmlDataStore. The DataStore's AutoSave property is currently set to True which means that, whenever a scheduled item is added, updated, or deleted, the records contained in the DataSet are written to an XML file. The path and name of the XML file are specified in the FileName property. The default value for the FileName is "\events.xml". The "\" in the FileName is interpreted by the XmlDataStore as the root directory of the web application. For this application, change the value to "\Rides.xml".

Note that even with these properties set to automatically persist the records, an error will occur if you run the application. That is because the application runs under the auspices of the ASPNET user account and that account does not have access to the file system on your computer. Before we can run the application, ASPNET must be granted read and write permissions to C:\inetpub\wwwroot\CityMobility (i.e., the location of this sample application).

To grant permissions to ASPNET user, do the following:

1. Open Windows Explorer and go to the C:\inetpub\wwwroot\CityMobility directory.
2. Select the Tools→Options menu item and go to the View page.
3. In the Advanced Settings listbox of the View page, scroll down to the bottom.
4. Make sure the **Use simple file sharing option** is not checked.
5. Click the OK button.
6. Right click the C:\inetpub\wwwroot\CityMobility folder in Windows Explorer and choose the *Sharing and Security* item from the popup menu.
7. Go to the Security page.
8. If the ASPNET user is not listed, click the Add button.
9. In the textbox labeled "Enter the object names to select, enter the text **<computer name>\ASPNET** and click the OK button.
10. With ASPNET user selected, assign it Read and Write permissions in the Permissions listbox.
11. Click the OK button.

At this point, the application will automatically persist scheduled rides to the Rides.xml file. The next step is to configure the DayController so that the correct time period is displayed in the WebPlanner.

## ASP.NET 2.0 or higher

### Configuring the DataSource in ASP.NET2.0 or higher

The WebPlanner has few requirements regarding how the scheduled items are stored in the database. The requirements are as follows:

1. The scheduled events must be stored in a single table.
2. The scheduled events must be uniquely identified via some type of key field.
3. Each scheduled event must have a start time and an end time.

Create a new database table with at least a key field, a start time field and an end time field. For this table, create a new OleDbDataSource or SqlClientDataSource and connect the datasource to WebPlanner.DataSource. Set the WebPlanner.KeyField, WebPlanner.StartTimeField, WebPlanner.EndTimeField properties to the table fields used for this. When additional fields are used, define these as well, ie. WebPlanner.SubjectField, WebPlanner.NotesField, WebPlanner.ResourceField. When the built-in WebPlanner recurrency feature is used, 3 additional fields are required:

RecurrencyField : A string field for storing the WebPlanner recurrency formula

FirstStartTimeField : this field is a DateTime field holding the start time of the first event in the series. Note that StartTimeField now holds the start time of the first event in the recurrent series.

FirstEndTimeField : this field is a DateTime field holding the end time of the first event in the series. Note that EndTimeField now holds the end time of the last event in the recurrent series.

It is important that when defining the DataSource component, proper commands are generated for INSERT, UPDATE, DELETE. Visual Studio .NET 2005 can do this for you from the DataSource configuration dialog. In case a database autoincrement field is used, make sure that the INSERT command will not insert the key field and that you have set WebPlanner.AutoIncrementKey to true.

## Configuring the Controller

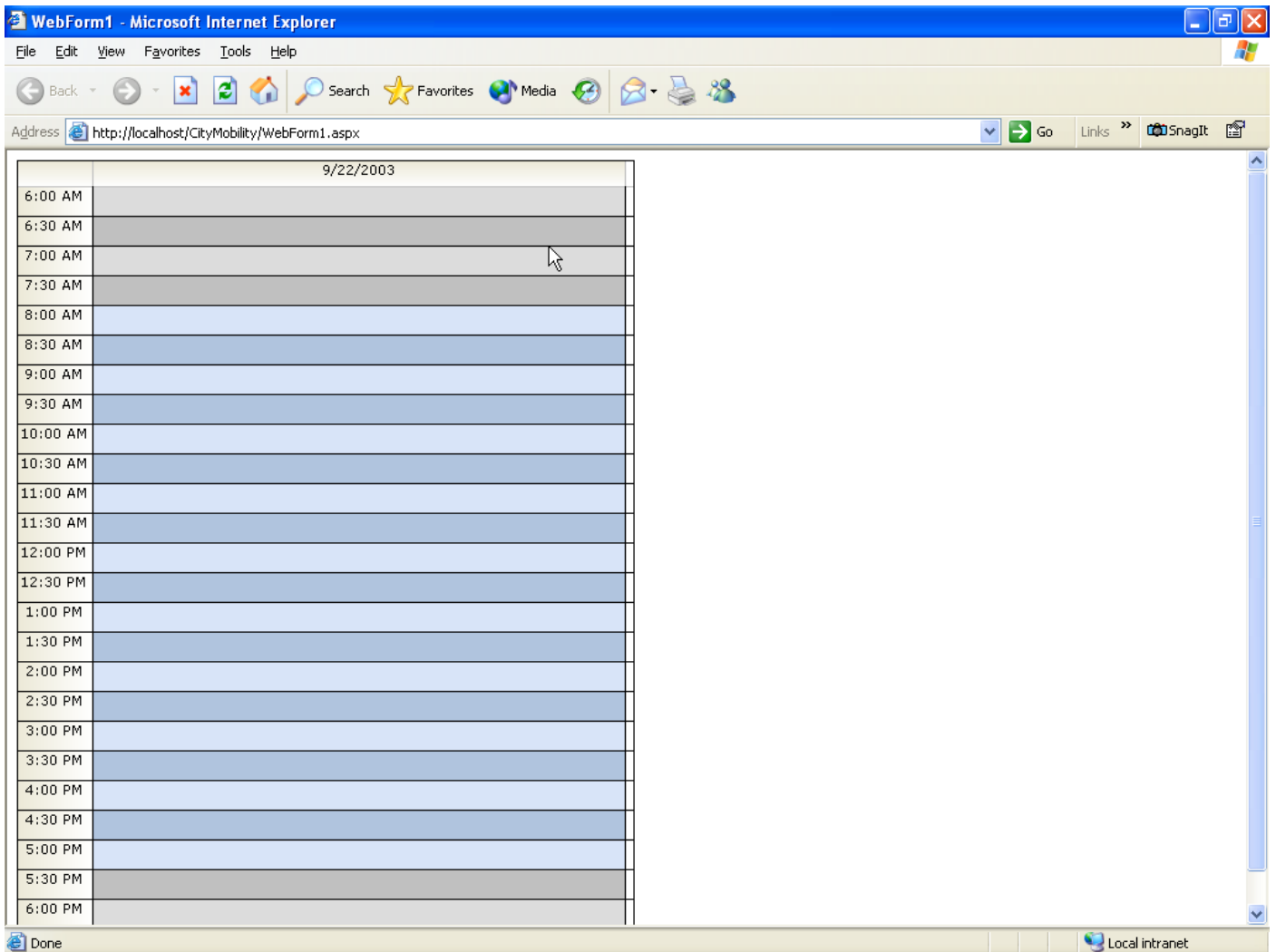
The WebPlanner is highly configurable, built to support seven standard modes (e.g., Day, Month, Week) as well as almost any custom modes a person can imagine. As a result, the mode-related properties available via the WebPlanner.Mode property are elaborate and can be confusing for people new to the WebPlanner. To quickly and easily configure a WebPlanner's mode, this product includes a number of controllers. In this project, the DayController is used to put the WebPlanner in Day mode. Day mode displays information for one or more days, broken down by hour and minute.

In the first section of this chapter, you hooked up the DayController to the WebPlanner via the WebPlanner's Controller property. In this section, the DayController will be configured to set the correct active zones for the CityMobility application. The active zone for bus rides is from 8:00 AM to 5:30 PM. This handles the majority of the needs of CityMobility customers and allows the bus drivers to get home at a half-way decent time.

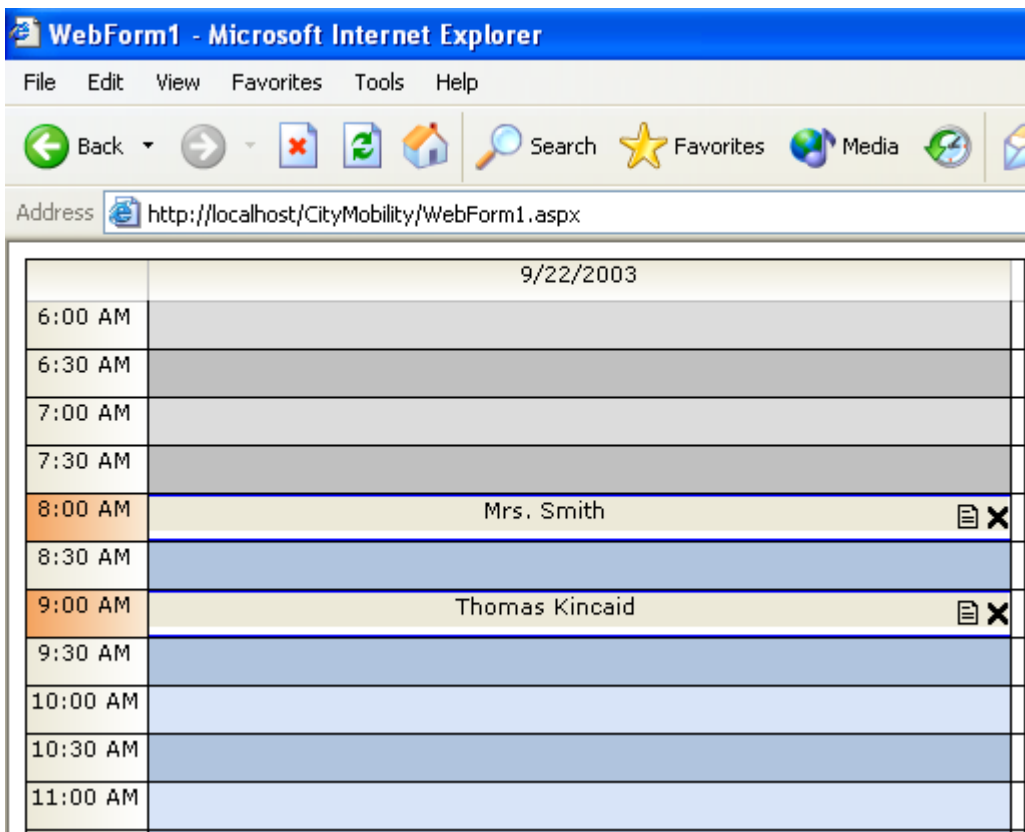
To change the active zone, do the following in the CityMobility project:

1. Select the DayController1 component.
2. On the Property page, set the ActiveStartTime to **8:00**. The ActiveStartTime property is of type TimeSpan. The time is specified in 24 hour format (e.g., 3:00 PM is 15:00).
3. Set the ActiveEndTime to **17:30**.
4. By default, the WebPlanner displays time slots up to 10:30 PM for Day mode. This is more than CityMobility needs to display, so set the EndTime property of DayController1 to the value **19:00**.
5. Set the NumberOfDays to the value **1**. This is not tied to the active zone, but it will cause the WebPlanner to display bus rides for one day at a time instead of 7.

At this point, run the application. The web browser should look similar to the following:



Let's take CityMobility for a spin. Using the mouse, left click on the time slot for 8:00 AM. Click OK when prompted to enter an item. At this point, a window appears asking for the item's caption. Enter "Mrs. Smith" and click the OK button. Add another one for Thomas Kincaid at 9:00 AM. When finished, the web browser should show the following:



Congratulations! A bare-bones version of the CityMobility application is running! In the next section, you'll learn how to customize the WebPlanner so that it takes bus drivers into account and allows the user to move from one day to the next.

## Customizing the WebPlanner

Now that the WebPlanner has been configured and it is able to save the scheduled bus rides, it is time to add some features that will make the application useful. The following sections show you how to have the WebPlanner switch its view from one day to another, how to configure the resources to which scheduled events are assigned, and how to use Day Mappings to alter the organization of the information displayed by the WebPlanner.

### Controlling the date

Typically, the user of a scheduling application not only needs to know what is being scheduled for today, they also need to know what openings are available in the following days. CityMobility, the organization for which this application is being created, requires its customers to schedule bus rides one week in advance. So when the application first loads, it should show scheduled rides for the date that is one week from now. The user should also be able to scan to the next day or to the previous day. If a customer calls to inquire about a bus ride scheduled for today, the user should be able to click a button that shows the rides scheduled for today.

To implement these requirements, do the following:

1. Relocate the WebPlanner so that there is room to place several buttons above it.
2. Add 4 buttons and set their properties as follows:

Button name	Button text
buttonPrev	"< Prev"
buttonNext	"Next >"

buttonToday	"Today"
buttonNextWeek	"Next Week"

3. Specify the following handlers for the Click event of each button.

### C# Event handlers

```
private void buttonPrev_Click(object sender, System.EventArgs e)
{
    DayController1.Prev();
}

private void buttonNext_Click(object sender, System.EventArgs e)
{
    DayController1.Next();
}

private void buttonToday_Click(object sender, System.EventArgs e)
{
    DayController1.Date = DateTime.Today;
}

private void buttonNextWeek_Click(object sender, System.EventArgs e)
{
    DayController1.Date = DateTime.Today.AddDays(7);
}
```

### VB.NET event handlers

```
Private Sub buttonPrev_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles buttonPrev.Click
    Me.DayController1.Prev()
End Sub

Private Sub buttonNext_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles buttonNext.Click
    Me.DayController1.Next()
End Sub

Private Sub buttonToday_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles buttonToday.Click
    Me.DayController1.Date = DateTime.Today
End Sub

Private Sub buttonNextWeek_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles buttonNextWeek.Click
    Me.DayController1.Date = DateTime.Today.AddDays(7)
End Sub
```

4. Add the following code to the form's Page\_Load method:

### C# Page\_Load

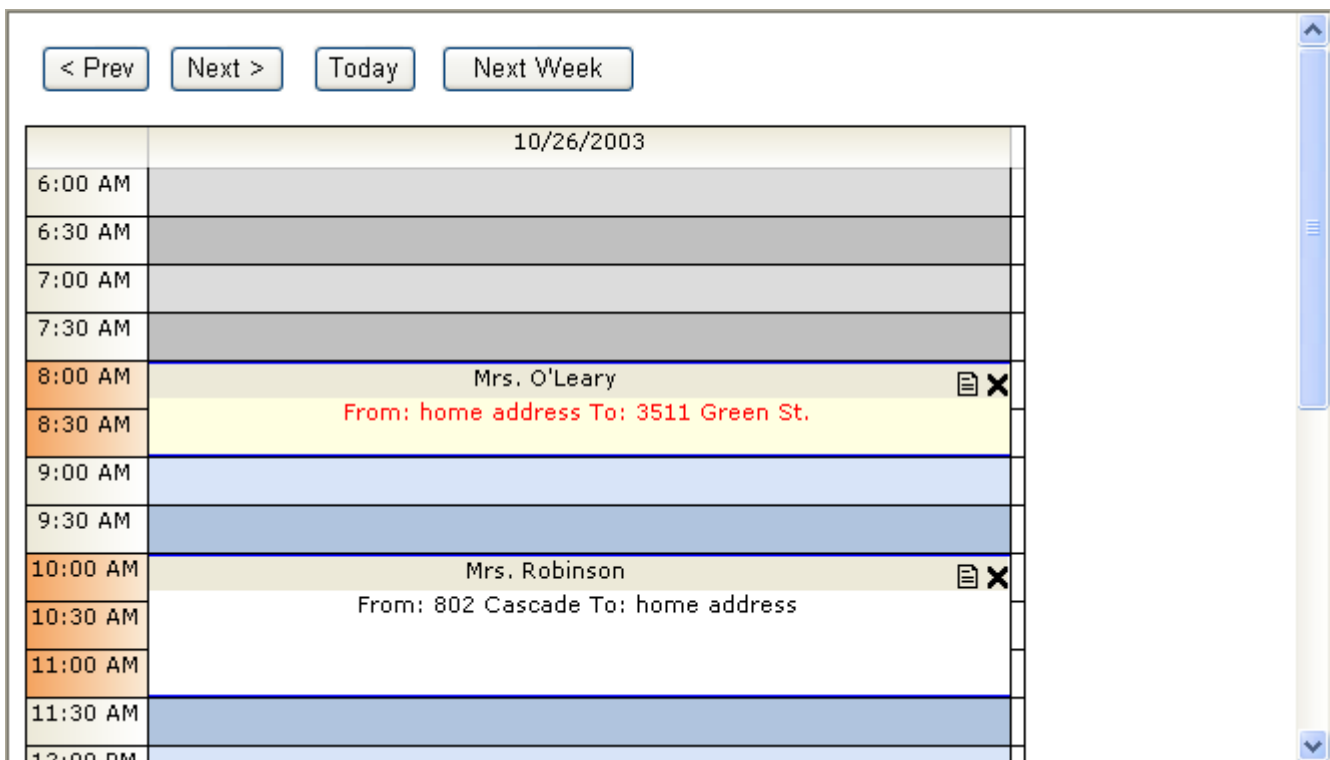
```
private void Page_Load(object sender, System.EventArgs e)
{
    if (!Page.IsPostBack)
        DayController1.Date = DateTime.Today.AddDays(7);
}
```

### VB.NET Page\_Load

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    If (Not Page.IsPostBack) Then
        DayController1.Date = DateTime.Today.AddDays(7)
    End If
End Sub
```

As shown in the code for the event handlers, the DayController contains several methods that make it easy to control the date for which the WebPlanner displays information. The Prev method moves the WebPlanner to the previous time period. In the case of the DayController, the time period is one day. The Next method moves to the next time period. If the WebPlanner should show events for a specific date, set the date via the DayController's Date property.

When executed, the application should look similar to the following:



Run the application and verify that the starting date is one week from today. Next, click the buttons to verify they cause the expected behavior.

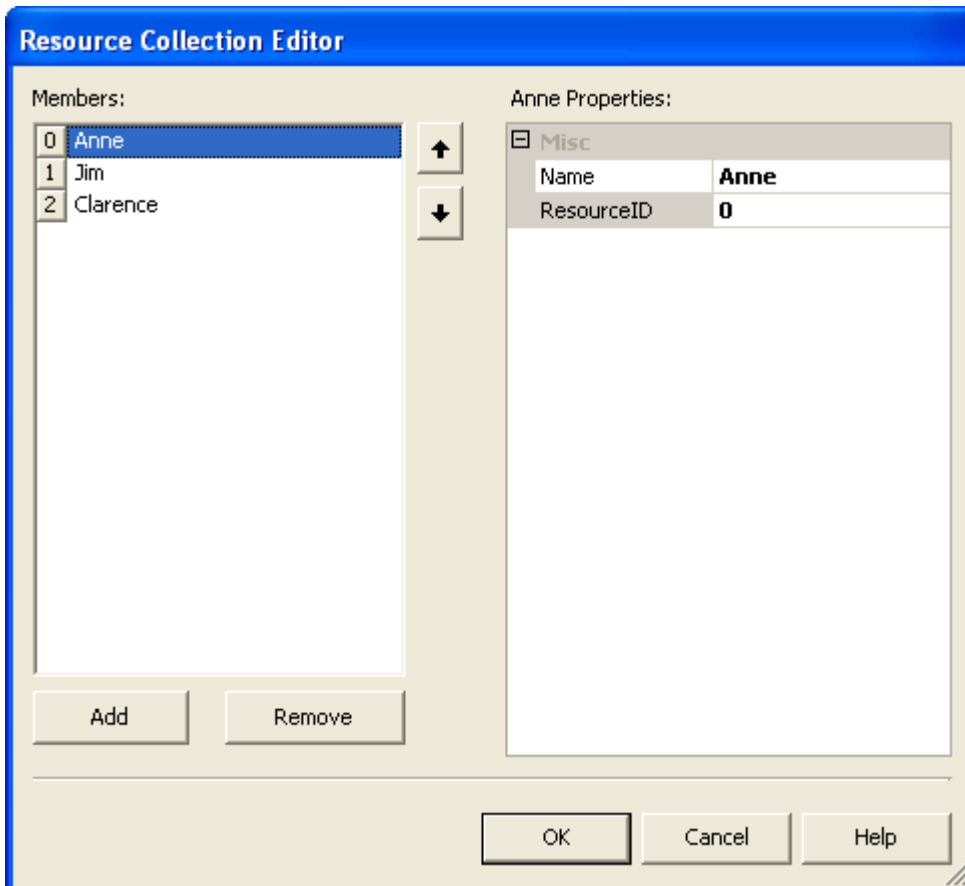
Specifying resources

CityMobility has three bus drivers that serve their customers. When a customer calls to schedule a ride, the dispatcher is responsible for assigning the ride to a bus driver having time to handle the trip on the requested day. It's now a requirement that the dispatcher be able to see the scheduled rides for each bus driver on any given day.

In WebPlanner terminology, each bus driver is considered to be a resource. Each event scheduled via the WebPlanner must be associated with a resource. Behind the scenes, the WebPlanner has already been associating each scheduled event with a default resource. Due to CityMobility's requirements, the resources must now be officially treated as bus drivers. A resource has a name and a unique ID. The name may be displayed on the WebPlanner so should be something meaningful. In this scenario, the resource name will be the name of the bus driver. The unique ID is an integer value (see section [Configuring the DataStore](#)) and is used to tie the scheduled items to a resource.

Resources can be specified at design time or programmatically at run time. For CityMobility, we will pretend the turnover for bus drivers is extremely low and specify the bus drivers at design time. To set up the bus drivers in the WebPlanner, do the following:

1. In the IDE, click on the WebPlanner and go to the Properties page.
2. In the Properties page, open the collection editor for the WebPlanner's Resources property.
3. Add three resources to the Resources collection editor.
4. Assign the following values to the Name property of the resources: Anne, Jim, and Clarence. These are the names of the bus drivers. When you have finished, the collection editor should look similar to the following:



At this point, the display of the WebPlanner is still showing bus rides for the first resource who is now named Anne. We need it to show the bus rides for each driver. To have it do so, do the following:

1. In the IDE, click the DayController and go to the Properties page.
2. In the Properties page, change the value of the DayMapping property to **MultiResource**.
3. Run the application. The application should now display one column per resource as shown in the following screen shot.



	Anne	Jim	Clarence
6:00 AM			
6:30 AM			
7:00 AM			
7:30 AM			
8:00 AM	Mrs. O'Leary		
8:30 AM	From: home address To: 3511 Green St.		
9:00 AM			
9:30 AM			
10:00 AM	Mrs. Robinson		
10:30 AM	From: 802 Cascade To: home address		
11:00 AM			
11:30 AM			
12:00 PM			

When you add a bus ride to Jim's column, the WebPlanner will automatically associate it with Jim. Bus rides scheduled in Clarence's column will be tied to Clarence, and so on. If you drag a bus ride from Clarence's column to Anne's column, the WebPlanner re-associates the ride with Anne.

The DayMapping property allows you to show scheduled events organized by day, by resource, and even grouped by resource for multiple days or grouped by day for multiple resources. For more information about DayMapping, see the Day section which talks about Day mode and the DayController.

At this point, try adding bus rides for each of the bus drivers and scanning through the week using the buttons at page top. One glaring omission you may notice is that the WebPlanner no longer displays the date. To fix this problem, add a Label next to the Next Week button. Add an event handler for the WebPlanner's PreRender event and, using the following example as a guideline, add to it code to update the label's text.

**C# Page\_Load**

```
private void WebPlanner1_PreRender(object sender, System.EventArgs e)
{
    Label1.Text = WebPlanner1.Controller.Date.ToString();
}
```

**VB.NET Page\_Load**

```
Private Sub webPlanner_PreRender(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles webPlanner.PreRender

    Label1.Text = WebPlanner1.Controller.Date.ToString()

End Sub
```

At this point, you have created a very basic scheduling application for the CityMobility organization. It shows the bus rides scheduled per bus driver on any given date. For more information about the WebPlanner's capabilities, please read the next chapter which discusses the WebPlanner in detail.

# WebPlanner

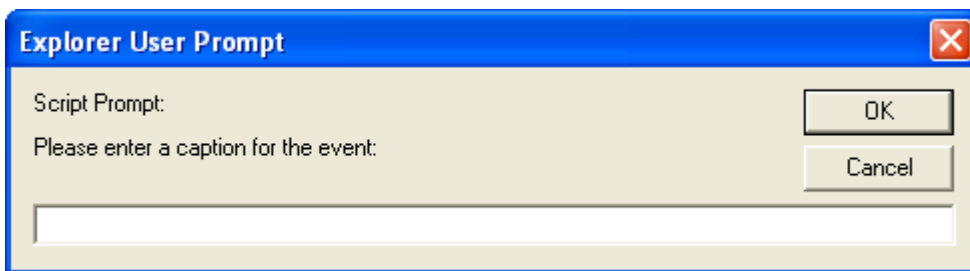
The WebPlanner provides comprehensive support for scheduling and resource planning applications. One way to look at the WebPlanner is to consider it an engine driven by a [Controller](#) component. The fuel for the engine is typically provided by a [DataStore](#) component in ASP.NET 1.1 and the DataSource in ASP.NET 2.0. This chapter discusses the various parts of the engine.

## Create and Edit Events

Users can create new events in any of the following ways:

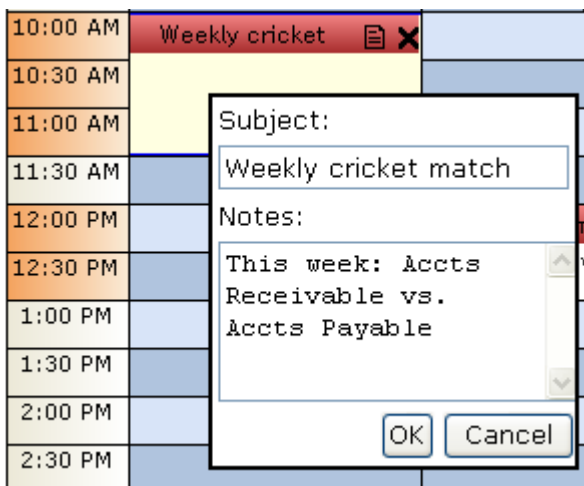
- Clicking the Insert glyph (i.e., '+') in the header of a day cell.
- Clicking on a day cell.
- Selecting a contiguous range of day cells.

The default behavior of the WebPlanner is to display a client-side dialog built into the browser. The dialog asks the user to enter the caption of the event.



Once the user has created the event, they may then enter its notes by editing the event. The WebPlanner has two built-in edit modes: Popup editing and In Place editing. Two custom modes are also available but they require you to perform additional work. For more information about those modes, see the [Custom Editing](#) section of this chapter.

To change the editing mode, use the WebPlanner's **EditType** property. To place the planner in Popup Edit mode, set its value to **PopupEdit**. In this mode, the user edits an event by clicking on the edit glyph displayed in the event's caption, by clicking the selected event once, or by clicking an event twice (once to select it, once to edit it). In all three cases, the WebPlanner displays a popup edit window.



Clicking the OK button saves the changes to the event. Clicking cancel closes the edit window. Use the **WebPlanner.PopupEdit** composite property to control the colors, font, and labels displayed in the popup edit window.

If the WebPlanner's EditType is set to **InPlace**, the event notes can be edited by clicking in the body of the event. The body turns into an edit window with a vertical scroll bar. The text can be changed in place. To save the changes, click the checkmark in the event's caption. To cancel the changes, click the "X".

9:30 AM	
10:00 AM	Weekly cricket ✓ X
10:30 AM	This week, Accts Receivable vs. Accts
11:00 AM	Payable
11:30 AM	

When in InPlace editing mode, right click on the caption in order to change it. The WebPlanner displays the same edit dialog shown at the beginning of this section. The default dialogs and editing behaviors may not be suitable for some applications. The WebPlanner has two custom edit modes that allow you to insert your own ASP.NET web forms in place of WebPlanner's dialogs and edit windows. For more information on this topic, please read the following section.

## Custom Editing

The dialogs used by the WebPlanner for creating and editing events are very basic. They have a simple look and allow input of an event's caption and notes only. Many applications will find the need to collect additional information and/or enforce certain business rules related to the scheduling of events. The WebPlanner provides a lot of flexibility in this area. Using the EditType property, an application can place the planner into one of two custom editing modes. The modes are described in the following sections.

### Detail Edit mode

Use Detail Edit mode when you want your user to create or edit events using your own ASP.NET web form instead of the default popup. For example, the user may need to enter application-specific information in addition to the meeting's topic or notes. Do not use this mode if other types of actions are to be performed when the user selects cells. Instead, use [Custom mode](#).

Two types of detail editing are possible. When EditType is set to DetailEdit, a popup window showing the form to perform the editing is used. When EditType is set to DetailEditEmbedded, the form for editing is shown inside a dialog inside the same browser window as the WebPlanner. The visual appearance of this dialog can be further customized with the properties WebPlanner.DetailEdit.EmbeddedBackColor, WebPlanner.DetailEdit.EmbeddedCaptionColor, WebPlanner.DetailEdit.EmbeddedCaptionForeColor.

**Note:** To see an excellent example of this mode, look at the Meeting Rooms portion of the example application installed with WebPlanner. The example application is installed in the <WebPlanner root>\Examples directory.

To place the WebPlanner in Detail Edit mode, set its **EditType** property to the value **DetailEdit**. When the user attempts to create a new event or edit an existing event, the planner opens the ASP.NET web form identified via the **WebPlanner.DetailEdit.EditTarget** property. If the user attempts to view a read-only event, the planner opens the ASP.NET web form identified via the **WebPlanner.DetailEdit.ViewTarget** property. For this mode to function correctly, those properties must be filled with the relative or absolute URL of a web form.

In order for Detail Edit mode to work properly, the WebPlanner's **SelectionMode** property must be set to the value **SingleSelectAutoCreate** or **MultiSelectAutoCreate**.

The custom edit form is responsible for carrying out any required actions, saving or updating data in the application's database, and notifying the planner when the edit form is closing. The edit form can determine what it needs to do by looking at the parameters passed to it in the `HttpRequest`. When the form is invoked, the planner places several parameters in the `HttpRequest`. They provide enough information for the form to determine where a new event is to be placed or what event is being edited. The following table identifies the parameters:

Parameter name	Value
DetailType	Identifies the type of action being performed by the user. Values are "Create", "Edit", and "View".
EventKey	When editing or viewing a <code>PlannerEvent</code> , the value of this parameter is the <code>KeyID</code> (i.e., unique ID) of the event.
SelEnd	When adding a new event, the ending cell selected by the user. This is a base zero column number.
SelPos	The base zero row number selected by the user.
SelStart	When adding a new event, the beginning cell selected by the user. This is a base zero column number.

So that every developer does not have to spend time writing code to retrieve these parameters, `WebPlanner` includes a utility class that retrieves the parameters for you. The class is named **DetailUtil** and it has a method named **GetParameters**. If the application passes the page's `HttpRequest` (i.e., `Page.Request`) to the `GetParameters` method, that method will return an instance of class **DetailEditParameters**. The `DetailEditParameters` class has a property for each of the aforementioned parameters.

When creating a new event, the custom edit form needs to translate the `SelEnd`, `SelPos`, and `SelStart` parameters into meaningful `DateTime` values. The `WebPlannerUtil` class provides a **CellToDateTime** method that converts a row number and column number to a `DateTime` value.

When the custom edit form is ready to close, it should cause the invoking `WebPlanner` to refresh itself. The refresh forces the `WebPlanner` to reload its events and the new event or changes to an existing event will appear. To cause the refresh, use the **DetailUtil.RefreshParentWindow** method. This method registers a client-side script that tells the edit form's parent window to refresh itself and then closes the edit form.

The following code examples show how a custom edit form obtains the Detail Edit parameters:

### C# Example

```
private void InitDetails()
{
    ...
    // Grab the detail edit parameters out of the request.
    DetailEditParameters parameters = DetailUtil.GetParameters(Request);
    this.SelEnd = parameters.SelectionEnd;
    this.SelPos = parameters.SelectionPosition;
    this.SelStart = parameters.SelectionStart;

    Meeting meeting = null;
    if (parameters.EventKey > 0)
    {
        meeting = new Meeting(parameters.EventKey);
        this.Meeting = meeting;
    }
    ...
}
```

### VB.NET Example

```

Private Sub InitDetails()
    ...
    ' Grab the detail edit parameters out of the request.
    Dim parameters As DetailEditParameters = DetailUtil.GetParameters(Request)
    Me.SelEnd = parameters.SelectionEnd
    Me.SelPos = parameters.SelectionPosition
    Me.SelStart = parameters.SelectionStart

    Dim meeting As Meeting = Nothing
    If parameters.EventKey > 0 Then
        meeting = New Meeting(parameters.EventKey)
        Me.Meeting = meeting
    End If
    ...
End Sub

```

The following code example shows how the Save and Cancel buttons on a custom edit form use the `WebPlannerUtil.CellDateToTime` and `DetailUtil.RefreshParentWindow` methods. Note that this source code is taken from the Meeting Rooms example mentioned earlier in this section.

### C# Example

```

private void buttonSave_Click(object sender, System.EventArgs e)
{
    // Perform actions required to save the event.
    Meeting meeting = this.Meeting;
    if (meeting == null)
    {
        #region Add

        // Get the planner.
        PlannerBucket bucket = this.GetPlanner();

        // Create the meeting
        meeting = new Meeting();
        meeting.Topic = textBoxTopic.Text;
        meeting.Notes = textBoxNotes.Text;
        meeting.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue);
        meeting.CreatorID = this.CreatorID;

        if (bucket.WebPlanner != null)
        {
            meeting.StartTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner,
                this.SelStart, this.SelPos, true);
            meeting.EndTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner,
                this.SelEnd, this.SelPos, false);
        }
        else
        {
            meeting.StartTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner,
                this.SelPos, this.SelStart);
            meeting.EndTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner,
                this.SelPos, this.SelEnd);
        }
        meeting.Insert();

        #endregion Add
    }
    else
    {

```

```

#region Edit

meeting.Topic = textBoxTopic.Text;
meeting.Notes = textBoxNotes.Text;
meeting.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue);
meeting.Update();

#endregion Edit
}

// Register a start up script that refreshes the
// main window & closes this window.
DetailUtil.RefreshParentWindow(Page, true);
}

private void buttonCancel_Click(object sender, System.EventArgs e)
{
    // Force parent page to refresh. In the case of adding a new
    // meeting, this will deselect the selected cells.
    DetailUtil.RefreshParentWindow(Page, true);
}

```

## VB.NET Example

```

Private Sub buttonSave_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles buttonSave.Click

    ' Perform actions required to save the event.
    Dim mtg As Meeting = Me.Meeting
    If (Meeting Is Nothing) Then
        ' Get the planner.
        Dim bucket As PlannerBucket = Me.GetPlanner()

        ' Create the meeting
        mtg = New Meeting
        mtg.Topic = textBoxTopic.Text
        mtg.Notes = textBoxNotes.Text
        mtg.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue)
        mtg.CreatorID = Me.CreatorID

        If (Not bucket.WebPlanner Is Nothing) Then
            mtg.StartTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner, _
                Me.SelStart, Me.SelPos, True)
            mtg.EndTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner, _
                Me.SelEnd, Me.SelPos, False)
        Else
            mtg.StartTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner, _
                Me.SelPos, Me.SelStart)
            mtg.EndTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner, _
                Me.SelPos, Me.SelEnd)
        End If
        mtg.Insert()

    Else
        mtg.Topic = textBoxTopic.Text
        mtg.Notes = textBoxNotes.Text
        mtg.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue)
        mtg.Update()
    End If

```

```

' Register a start up script that refreshes the
' main window & closes this window.
DetailUtil.RefreshParentWindow(Page, True)

End Sub

Private Sub buttonCancel_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles buttonCancel.Click

' Force parent page to refresh. In the case of adding a new
' meeting, this will deselect the selected cells.
DetailUtil.RefreshParentWindow(Page, True)

End Sub

```

## Custom Mode

Custom mode is very similar to [Detail Edit mode](#). The main difference is that it does not automatically invoke the custom edit form specified in the WebPlanner's **DetailEdit** property. Instead, the application must invoke the form programmatically. An example of doing so is provided at the end of this section.

To put the WebPlanner into custom mode, do the following:

- Set the WebPlanner's **EditType** property to **Custom**.
- Set the WebPlanner's **SelectionMode** property to **SingleSelect** or **MultiSelect**.

In custom mode, the application knows that the user wishes to create an event when the WebPlanner's **CellsSelected** event is raised. If the SelectionMode is SingleSelect, the CellsSelected event is raised when the user has clicked on a single cell in the WebPlanner. If the SelectionMode is MultiSelect, the event is raised after the user selects one or more cells.

The application knows that the user wants to edit an event when the WebPlanner's **EventCustomEdit** event is raised. WebPlanner raises EventCustomEdit when the user clicks on an edit glyph in the event's caption, when the user clicks the currently selected event, or when the user double clicks an event (the first click selects, the second raises the event).

The application is responsible for passing the appropriate parameters to and invoking the custom edit form. The custom edit form must then retrieve the parameters, configure its interface, and save any changes made by the user to the database. When the user has saved or canceled their changes, the edit form must refresh its parent window and close itself using the DetailUtil.RefreshParentWindow method. Doing so causes the WebPlanner to refresh and reload the events for the currently displayed month.

The following code example is from an existing application and shows how can invoke its custom edit form. Note that some of the code, such as the CustomPrincipal class, is specific to the application.

### C# Example

```

// Called when the user is creating an event.
private void webPlanner_CellsSelected(object sender,
    TMS.WebPlanner.PlannerCellsSelectedEventArgs e)
{
    // Create an event for the selected cells.
    PlannerEvent plannerEvent = webPlanner.CreateEventAtSelection();

```

```

// Add information about the meeting to the session. It will be
// picked up by the popup window.

Session.Add("PlannerEvent", plannerEvent);
Session.Add("RoomID", e.SelectedResource.ResourceID);

CustomPrincipal principal = (CustomPrincipal) Page.User;
if (principal != null)
    Session.Add("CreatorID", principal.Person.ID);
else
    throw new CustomException("No principal found when creating a new event.");

// Add startup script that displays a popup to capture the meeting details.
string script = "<script language='javascript'> " +
    "window.open('DetailPopup.aspx', 'AddMeeting'," +
    "'width=400, height=350, menubar=no, center=yes, resizable=no');" +
    "</script>";
Page.RegisterStartupScript("AddMeeting", script);
}

// Called when the user is editing an event.
private void webPlanner_EventCustomEdit(object sender,
    TMS.WebPlanner.PlannerEventClickEventArgs e)
{
    // Add information about the meeting to the session. It will be
    // picked up by the popup window.
    Session.Add("RoomID", e.Event.ResourceID);
    Session.Add("MeetingID", e.Event.Key);
    CustomPrincipal principal = (CustomPrincipal) Page.User;
    if (principal != null)
        Session.Add("CreatorID", principal.Person.ID);
    else
        throw new CustomException("No principal found when creating a new event.");

    // Add startup script that displays a popup to capture the meeting details.
    string script = "<script language='javascript'> " +
        "window.open('DetailPopup.aspx', 'EditMeeting'," +
        "'width=400, height=350, menubar=no, center=yes, resizable=no');" +
        "</script>";
    Page.RegisterStartupScript("AddMeeting", script);
}

```

## VB.NET Example

```

' Called when the user is creating an event.
Private Sub webPlanner_CellsSelected(sender As Object, _
    e As TMS.WebPlanner.PlannerCellsSelectedEventArgs)

    ' Create an event for the selected cells.
    Dim plannerEvent As PlannerEvent = webPlanner.CreateEventAtSelection()

    ' Add information about the meeting to the session. It will be
    ' picked up by the popup window.
    Session.Add("PlannerEvent", plannerEvent)
    Session.Add("RoomID", e.SelectedResource.ResourceID)

    Dim principal As CustomPrincipal = CType(Page.User, CustomPrincipal)
    If Not (principal Is Nothing) Then
        Session.Add("CreatorID", principal.Person.ID)
    }

```



```

Else
    Throw New CustomException("No principal found when creating a new event.")
End If
' Add startup script that displays a popup to capture the meeting details.
Dim script As String = "<script language='javascript'> " + _
    "window.open('DetailPopup.aspx', 'AddMeeting'," + _
    "'width=400, height=350, menubar=no, center=yes, resizable=no');" + _
    "</script>"
Page.RegisterStartupScript("AddMeeting", script)

End Sub

' Called when the user is editing an event.
Private Sub webPlanner_EventCustomEdit(sender As Object, _
    e As TMS.WebPlanner.PlannerEventClickEventArgs)

    ' Add information about the meeting to the session. It will be
    ' picked up by the popup window.
    Session.Add("RoomID", e.Event.ResourceID)
    Session.Add("MeetingID", e.Event.Key)
    Dim principal As CustomPrincipal = CType(Page.User, CustomPrincipal)
    If Not (principal Is Nothing) Then
        Session.Add("CreatorID", principal.Person.ID)
    Else
        Throw New CustomException("No principal found when creating a new event.")
    End If
    ' Add startup script that displays a popup to capture the meeting details.
    Dim script As String = "<script language='javascript'> " + _
        "window.open('DetailPopup.aspx', 'EditMeeting'," + _
        "'width=400, height=350, menubar=no, center=yes, resizable=no');" + _
        "</script>"
    Page.RegisterStartupScript("AddMeeting", script)

End Sub

```

## SideBar

The time axis of the WebPlanner is represented by a sidebar. The properties of the sidebar are controlled via the **PlannerSideBar** class and the WebPlanner's sidebar may be accessed via the **WebPlanner.SideBar** property. By default, the sidebar is visible and displayed on the left hand side of the WebPlanner. The application may hide the sidebar entirely via the **PlannerSideBar.Visible** property. To reposition the sidebar to the top, right, or left and right of the WebPlanner, use the **PlannerSideBar.Position** property.

The sidebar is divided into time slots. The meaning of each time slot is determined by the WebPlanner's [mode](#). For example, the time slots for Day mode represents minutes and hours of the day. The time slots for Month mode represent a day of the month. By default, the sidebar will render time information in each of its time slots. For example, the following screenshot shows the half day periods rendered for Half Day mode.

	Truck A114	
10/26/2003 AM		
10/26/2003 PM		
10/27/2003 AM		
10/27/2003 PM		
10/28/2003 AM		
10/28/2003 PM		

To prevent the sidebar from rendering time information in every time slot, use the **PlannerSideBar.TextInterval** property. When set to the value 0 or 1, information is rendered in each cell. When the value is changed to 2, text is rendered in every 2nd cell. A value of 3 renders text in every 3rd cell, and so on. The following screenshot shows the result of specifying a value of 2 for the TextInterval property.

	Truck A114	
10/26/2003 AM		
10/27/2003 AM		
10/28/2003 AM		

The width of the sidebar is controlled via the **Width** property. The height of each time slot matches the value specified for the **PlannerDisplay.RowHeight** property.

The background color of the sidebar is controlled via the **BackColor** and **BackColorTo** properties. If the colors are identical then the time slots have a solid background. If the colors differ then the background is rendered as a gradient. The gradient direction is controlled via the **GradientDirection** property. If one or more cells corresponding to the time slot are occupied by a **PlannerEvent**, the sidebar renders its background using the **OccupiedColor** and **OccupiedColorTo** properties. To turn off special coloring for occupied time slots, set the **ShowOccupied** property to the value **false**. The following screenshot shows an example of time slots being drawn in their occupied color.

	Truck A114	
10/26/2003 AM		
10/27/2003 AM	Hancock	☰ ✕
10/28/2003 AM	Stilsworth	☰ ✕

## Header

The header describes the positions and groups displayed in the WebPlanner. For example, if the planner is in Day mode with MultiDay mapping, the header displays the date for each position. If the positions are grouped by resource then the header displays the name of the resource corresponding to the group. To access the header-specific properties, expand the WebPlanner's **Header** property. The header may be disabled by setting the **Visible** property to false.

By default, when the [sidebar](#) is positioned to the left and/or right of the planner, the header is rendered on the top of the planner. When the sidebar is positioned on the top of the planner, the header is rendered to the left. The header's position may be controlled via the header's **Position** property. Using this property, the header may be rendered at its default location, the opposite location (e.g., the bottom of the WebPlanner), or both.

To control the coloring of the header, use the **BackColor** and **BackColorTo** properties. If both properties contain the same color then the background is rendered as one solid color. For a gradient effect, assign different values to the properties. To control the direction in which the gradient is drawn, use the **GradientDirection** property.

Use the **LineColor** property to control the color of the cell borders rendered in the header. Use the **ForeColor** property to specify the color of the header text.

The following screenshot points out landmarks within the header:

	7/16/2004		
	Dr. Robinson	Dr. Atkinson	Dr. Johnson
7:00 AM			

### 1: SideBar space

The header renders a blank area over the top of the [sidebar](#). To render text in this area, specify a value for the WebPlanner's **TopLeftCell** property.

### 2: Position caption

When the **Header.UpdateCaptions** property is set to true, the WebPlanner automatically generates captions for each position. In this screenshot, the planner is in [Day mode](#) with [MultiDayResource](#) day mapping. Each position displays the name of a [resource](#). If Header.UpdateCaptions property is set to false, the application is responsible for updating the **Caption** property of each Position in the [Positions](#) collection.

An application may detect user clicks within a position by specifying a delegate (e.g., event handler) for the **HeaderClick** event. When a delegate is specified, the header captions are rendered as hyperlinks. The arguments passed to the delegate indicate the index of the position and the resource associated with the position.

### 3: Group caption

When the planner is grouping positions and the **Header.UpdateCaptions** property is set to true, the WebPlanner automatically generates captions for each group. In this screenshot, each group bears the date. If the **Header.UpdateCaptions** property is set to false, the application is responsible for adding group captions to the WebPlanner's **GroupCaptions** property.

### 4: Header item placeholder

PlannerEvents may be inserted into the header under certain conditions. The header renders a blank area that serves as a placeholder for header events. The height of the area is equal to the **Header.Height** property.

## Header Groups and SubGroups

In addition to the WebPlanner header, it is also possible to add custom groups and subgroups. Groups and Subgroups can be used to indicate positions that belong together, for example doctors in different hospital and/or different departments. Groups and SubGroups operate separately from the header and are set only by the properties **WebPlanner.Header.Groups** and **WebPlanner.Header.SubGroups**. These are collections that hold HeaderGroup objects with which the caption text for a group can be set as well as the span. The span is the number of positions that a group or subgroup spans. When setting the Groups or SubGroups, make sure that the total value of the span of positions is equal to the nr. of positions in the WebPlanner.

Example:

In a WebPlanner with 8 positions, the WebPlanner should show a group Cars of 4 positions, a group Trucks of 2 positions and a group Sportcars of 2 positions. This can be configured in the WebPlanner with the code:

```
WebPlanner1.Header.Groups.Add(new TMS.WebPlanner.HeaderGroup("Cars",4));
WebPlanner1.Header.Groups.Add(new TMS.WebPlanner.HeaderGroup("Trucks",2));
WebPlanner1.Header.Groups.Add(new TMS.WebPlanner.HeaderGroup("Sportcars",2));
```

## Bands and Zones

When the WebPlanner draws its grid, by default it uses the **Bands** property to alternate the coloring of the cells. If the **Bands.Visible** property is set to false then the cells within the active and inactive time zones are rendered using one color per zone. The Bands property defines four banding colors:

- ActivePrimary
- ActiveSecondary
- InactivePrimary
- InactiveSecondary

The Active colors are used to render the background of the active time zone or days marked as active via the **InactiveDays** property. The Inactive colors are used to render the background of the inactive time zones or inactive

days. In Day mode, the starting and ending cells for the active zone may be controlled through the [DayController](#). The following screenshot identifies the active and inactive colors and time zones for Day mode.

	9/22/2003
6:00 AM	
6:30 AM	
7:00 AM	
7:30 AM	
8:00 AM	
8:30 AM	
9:00 AM	
9:30 AM	
10:00 AM	
10:30 AM	
11:00 AM	
11:30 AM	
12:00 PM	
12:30 PM	
1:00 PM	
1:30 PM	
2:00 PM	
2:30 PM	
3:00 PM	
3:30 PM	
4:00 PM	
4:30 PM	
5:00 PM	
5:30 PM	
6:00 PM	

**1: Inactive Secondary**

The odd rows of an inactive zone are rendered in the InactiveSecondary color.

**2: Inactive Primary**

The even rows of an inactive zone are rendered in the InactivePrimary color.

**3: Leading Inactive Zone**

In Day mode, the active zone is preceded by an inactive zone. Note that the WebPlanner does not prevent the user from adding items to the inactive zone. To prevent users from adding events to the inactive zone, create a [PositionStyle](#) that only allows selection within the active zone or create a PlannerEvent for the zone with its

**Background** property set to the value true. The latter would require creating PlannerEvents for the leading and trailing inactive time zones for each day that is displayed.

#### 4: Active Secondary

The odd rows of the active zone are rendered in the ActiveSecondary color.

#### 5: Active Primary

The even rows of the active zone are rendered in the ActivePrimary color.

#### 6: Active Zone

For Day mode, the active zone sits in between two inactive zones. Using [Position Styles](#), the active zone may be reconfigured per position based upon application-specific criteria.

#### 7: Trailing Inactive Zone

The active zone is followed by an inactive zone.

The aforementioned coloring scheme applies to the Day and Timeline modes. In other modes, the cells of the time axis represent days instead of hours and minutes. In those modes, the Active and Inactive banding colors are used to color active and inactive days. A day is considered inactive if its corresponding flag in the **InactiveDays** property is set to the value true. By default, Saturday and Sunday are considered to be inactive days.

## Positions

*Position* is a generic term for a column or row rendered in the WebPlanner. When the sidebar is located on the top of the planner, a position is rendered as a row. When the sidebar is rendered to the left and/or right of the planner, a position is rendered as a column. A position may associated with a resource, a specific date, or a combination of the two. When the WebPlanner places its PlannerEvents on its grid, it places them in the correct position according to the current [mode](#). The mode matches the PlannerEvent to the correct position.

In general, positions are controlled via the **PositionCount** or **Positions** properties. PositionCount identifies the raw number of positions to be displayed in the planner. The Positions property is a collection of Position instances that identify the position's caption. The two properties are interconnected. When the PositionCount changes, it adds positions to or removes positions from the Positions collection so that its contents match the PositionCount. When the Positions collection is changed at design time, the PositionCount reflects the number of positions now in the collection.

To add positions, set the number of positions via the PositionCount property. If the application is responsible for setting the position captions displayed in the header (i.e., the **Header.UpdateCaptions** property is set to false) then set the caption of each position via the Positions collection.

In general, the responsibility for setting positions lies with the developer at design time or the application at run time. However, [Day mode](#) will dynamically adjust the number of positions based upon the value of its **DayMapping** property.

The **EventWidth** property can be used to set a fixed width for all PlannerEvents. The width of the Position containing the PlannerEvents is automatically adjusted. Please note: non-overlapping PlannerEvents positioned on a Position with other overlapping events will automatically be stretched to the full width of the Postion.

When the **AutoPositionWidth** property is set to true, the widths of the PlannerEvents and Positions are automatically adjusted proportionally according to the number of overlapping PlannerEvents per Postion, relative to the total Planner width.

## Resources

A *resource* represents a person or thing whose time is being scheduled. For example, a resource could be a meeting room, a doctor, a university professor, a rental car, or a flight simulator. Each `PlannerEvent` rendered in the WebPlanner is associated with a resource. Each resource has a unique ID and a name. The unique ID is used to associate a `PlannerEvent` with a resource. The name may be used to display the name of the resource in the [header](#).

The WebPlanner maintains a collection of resources in its **Resources** property. If a resource is not specified at design time or dynamically created at run time, the WebPlanner creates a default resource with an ID of zero and the name "Default Resource". The WebPlanner does not provide any support for retrieving resources from a database or [data store](#). The responsibility for maintaining the resources lies with the developer.

Depending upon the [mode](#), the [positions](#) rendered in the WebPlanner are associated with a resource. For example, in [Month mode](#) each position is associated with a specific resource. In general, `Positions[0]` corresponds to `Resources[0]`, `Positions[1]` to `Resources[1]`, and so on. Even though the mode may have its positions tied to resources, the developer is responsible for both making sure the correct resources are in the Resources collection and the correct number of positions have been specified via the **PositionCount** or **Positions** properties.

Note that when a data store retrieves `PlannerEvents`, its criteria for doing so includes a start date and end date. The criteria do not include resource IDs. So it is possible for the data store to retrieve `PlannerEvents` for which there is no corresponding resource in the Resources collection. Such `PlannerEvents` are ignored.

## Default Event

The **DefaultEvent** property defines a `PlannerEvent` that is used as the model for all `PlannerEvents` instantiated via the [Items](#) collection editor and the [DataStores](#). For example, if you wish for all `PlannerEvents` to be read only then set the `DefaultEvent`'s **ReadOnly** property to the value **true**. For applications where `PlannerEvents` are added to the Items collection at design time, the coloring of the events may become out of synch with the colors used in the `DefaultEvent`. To update the colors of the items, use the "Apply default event colors" verb available in the design time environment.

If your application needs to programmatically create new `PlannerEvents`, it should clone the `DefaultEvent`. Use the WebPlanner's **CreateEvent** method to clone the `DefaultEvent` and add the new event to the Items collection.

## Items

The **Items** collection is the location in which the WebPlanner stores the `PlannerEvents` that are to be rendered. When the WebPlanner is connected to a [DataStore](#), this collection is refreshed upon each page request or postback. The logic for doing so is described in the following steps:

1. Remove all `PlannerEvents` whose **Origin** property is set to the value **ItemOrigin.DataStore**. This means that `PlannerEvents` programmatically added by the application are left in the collection. If the WebPlanner's **PreserveApplicationItems** property is set to the value **false** then all events are removed regardless of their origin.
2. Call the `DataStore`'s `Read` method to pull in the collection of events matching the current mode.
3. For each `PlannerEvent` retrieved by the `DataStore`, raise the **EventRetrieved** event. When all events have been retrieved, call the **EventsRetrieved** event.

Note that the Items collection may wind up containing events whose date and position fall outside the range displayed by the WebPlanner. In that case, those events are ignored but remain in the collection until the next page request or postback.

When the `DataStore` retrieves information for an event, it clones the WebPlanner's [DefaultEvent](#) property and modifies the clone's properties according to the retrieved information.

The application is free to modify the Items collection as necessary. More information about doing so can be found in the [Managing PlannerEvents by hand](#) section.

## Position Styles

When a [position](#) is rendered, its coloring is dictated by the [bands and zones](#). The result is that the coloring of positions is consistent from one position to the next. In certain situations, an application may need to change the coloring of a specific position, change the active time zone for the day, limit the area within which a user may schedule items, or even make the entire position read-only.

The solution for those cases is the `PositionStyle` class. The WebPlanner maintains a **PositionStyles** collection. By default, it is empty. However, the developer or application may add `PositionStyle` instances to the collection as needed and associated one or more positions with a position style. Each `PositionStyle` has a unique name. To associate a `Position` with the `PositionStyle`, do the following:

1. Find the position in the **Positions** collection.
2. Set the **Position.PositionStyle** property to the **Name** property of the `PositionStyle` that is to affect the rendering of the position.

Because it is referenced by name, a position style may be used by two or more positions.

The `PositionStyles` example installed with this product illustrates the use of position styles. Depending upon the number of appointments on a day, the cells for the day may be rendered in red (i.e., too many appointments for the day), yellow (the optimum number of appointments have been scheduled), or green (the optimum number of appointments has not been reached).

Note:

To further allow customization of the WebPlanner background colors, the delegate `SetCellColor` can be implemented. This delegate is called for the generation of every cell in the WebPlanner grid. The event argument parameters `Column`, `Row` indicate for which cell the color is queried. The color can be overridden by setting the event arguments `CellColor` property.

### Setting position colors

The **ColorActive** property of the `PositionStyle` specifies the color to be used in the position's active zone. Note that every cell in the active zone will be rendered in this color. There will be no banding. To retain the original banding colors, leave the value of the `ColorActive` property set to **Color.Empty**.

The **ColorInactive** property of the `PositionStyle` specifies the color to be used in the position's leading and trailing inactive zones. Note that every cell in the inactive zone will be rendered in this color. There will be no banding. To retain the original banding colors, leave the value of the `ColorInactive` property set to **Color.Empty**.

The **ColorNoSelect** property identifies the color of the cells within the no selection region. If the `ColorNoSelect` property is set to the value **Color.Empty** then it will be ignored and the original banding colors will be used.

### Adjusting the active zone

To adjust the active time zone for a position within [Day mode](#), set the `PositionStyle`'s **ActiveStart** and **ActiveEnd** properties. Both properties have an integer value representing the number of cells from the beginning of the planner's display. For example, a value of 16 for `ActiveStart` means that the active zone starts in the 16th cell from the beginning of the planner display. A value of 40 means the last cell of the active zone is the 40th cell from the start of the planner display.

To convert cell numbers to times, or vice versa, use the methods available in the `WebPlannerUtils` class.



## Establishing a no select zone

To prevent users from adding events outside the active time zone, the `PositionStyle` class provides the **MaxSelection** and **MinSelection** properties. The `MaxSelection` property identifies the last cell in the planner's display may be selected (and thus used to add a `PlannerEvent`). The `MinSelection` property identifies the first cell in the planner's display that may be selected. To restrict selection to the active time zone, set `MaxSelection` to the value of property **ActiveStart** and `MinSelection` to the value of **ActiveEnd**.

If the entire position is to be read only (e.g., a user may modify their events but not those events associated with other users) then set the `PositionStyle`'s **ReadOnly** property to the value `true`. Each position associated with the `PositionStyle` will be read-only for the current user.

## Layers

By default, all `PlannerEvents` retrieved for the current mode are displayed in the WebPlanner. The only caveat is that the `PlannerEvent`'s `ResourceID` matches a [resource](#) in the **Resources** collection. For most applications this is adequate. In other cases, users may wish to see a subset of the scheduled events based upon certain criteria. For example, each `PlannerEvent` may have a low, medium, or high priority and the user wishes to view only those with high priority. In this case, each priority is a *layer*.

The WebPlanner allows an application to set up arbitrary layers and assign `PlannerEvents` to those layers. For a hands on example of how this could be implemented, look at the Dr. Pepper's Office demo in the example application installed with this product. In the demo, appointments are assigned categories such as "Consultation", "Diagnostic", or "Routine Exam". Each appointment type is interpreted as a specific layer and the application allows the user to view specific types of appointments or combinations of appointment types.

Layers are established via the WebPlanner's **Layer** property and the `PlannerEvent`'s **Layer** property. Both properties are defined as type `System.int` but should be treated as bit flags. When the `WebPlanner.Layer` property has the value zero, it means "no layer is in effect". When the `PlannerEvent.Layer` property has the value zero, it means "this event does not belong to a layer".

The first layer is represented by the value 1, the second layer by 2, the third layer by 4, the fourth layer by 8, and so on. To assign a `PlannerEvent` to the 3rd layer, set its `Layer` property to the value 4. `PlannerEvents` may be assigned to more than one layer by ORing the bit values of the layers. For example, to assign a `PlannerEvent` to the 1st and 4th layers, set its `Layer` property to the value 9 (i.e.,  $1 + 8$ ). The `PlannerEvent`'s `Layer` property defaults to the value 0.

One of the best places to set the layer values for `PlannerEvents` is in the WebPlanner's **EventsRetrieved** event. When this event is raised, all events have been retrieved from the [data store](#). The delegate may walk through the collection of events specified in the event arguments and set the layer based upon whatever criteria are important.

By default, the WebPlanner's `Layer` property has the value zero. This means that the WebPlanner will render all `PlannerEvents` regardless of their layer. To have the WebPlanner show only those events assigned to the first layer, set the WebPlanner's `Layer` property to 1. To show the second layer, set it to 2, and so on. To show more than one layer, OR the layer values together. For example, to show the 1st and 4th layers, set the WebPlanner's `Layer` property to the value 9 (i.e.,  $1 + 8$ ).

## Keyboard Support

By default, the user interacts with the WebPlanner using only the mouse. To activate keyboard support, set the WebPlanner's **KeyboardEnabled** property to the value `true`. When keyboard support is enabled, the user can move from one cell to the next using the arrow keys. They can press the Insert key to create a new event and they can press the Delete key to delete the currently selected event.

## Client-side JavaScript

The WebPlanner uses a large chunk of client-side JavaScript to implement features such as drag and drop and the resizing of events. By default, this code is included in every page returned to the client browser. To reduce the download times of pages containing a WebPlanner control, the client-side JavaScript can be downloaded to the client once and cached. To enable this feature, do the following:

1. Set the Webplanner's **CachedScript** property to the value true.
2. Place a copy of the file webplanner.js in the directory `c:\inetpub\wwwroot\aspnet_client\cs_webplanner\2_0_0_0`

When the page is requested, the WebPlanner includes a script within the page's HTML. The script tells the client browser to download the webplanner.js file. Because the webplanner.js file is placed in the aspnet\_client directory structure, it may be shared among multiple web forms and web applications.

Note that you can place the webplanner.js file in a directory other than the one mentioned in the previous steps. To do so, set the WebPlanner's **CachedScriptPath** property to the directory location. Note that you must specify an absolute URL or a URL relative to your website.

## Modes and Controllers

---

The WebPlanner renders itself based upon its current *mode*. A mode describes both the units of time displayed along the time axis and the manner in which the PlannerEvents are organized into [positions](#). For example, in Month mode the units of time are days and each position represents a [resource](#). The WebPlanner has several native modes including Day, Day Period, Half Day Period, Month, MultiMonth, Week, and Timeline mode. It defaults to Day mode. For each mode there is a corresponding controller component. For example, the TimelineController configures the WebPlanner for Timeline mode. The WebPlanner's mode can be controlled either by connecting it to a controller or by configuring the appropriate properties in the WebPlanner.

To connect a WebPlanner to a controller, do the following:

1. In the IDE, drop the appropriate controller onto your form.
2. Click the WebPlanner and go to the Properties page.
3. On the Properties page, select the controller in the WebPlanner's **Controller** property.

To set a WebPlanner's mode without the aid of a controller, do the following:

1. In the IDE, click the WebPlanner and go to the Properties page.
2. On the Properties page, expand the WebPlanner's **Mode** property.
3. In the **Mode** property, select the appropriate mode via the **PlannerType** property.
4. If you chose "Day" for **PlannerType**, select the appropriate value for the Mode's **DayMapping** property.

**Note:** The DayMapping property of the WebPlanner's Mode only affects the WebPlanner when Day mode is active. It has no effect upon other modes.

Once the PlannerType and DayMapping have been set, other property values must be set so that the mode knows the period of time it must render, the number of positions, etc. Controllers do this automatically in a user-friendly manner. The following sections describe the purposes and capabilities of the modes and their corresponding controllers.

### Day

In Day mode, the time axis represents hours and minutes of one day. The active zone is a contiguous set of hours representing the portion of the day in which PlannerEvents may be scheduled. The active zone is preceded and followed by inactive zones representing non-work hours (i.e., time frames in which events should not be scheduled). The WebPlanner does not prevent the user from scheduling an event in the inactive time zone. To prevent the user from doing so, either use [PositionStyles](#) or fill each inactive time zone with a background PlannerEvent. The following screenshot shows a portion of a WebPlanner rendered in Day mode.

Dr. Robinson	
7:00 AM	
7:30 AM	
8:00 AM	
8:30 AM	
9:00 AM	

By default, the granularity of each time slot is 30 minutes. To change the granularity, expand the WebPlanner's **Display** property and change the value of the **TimeUnit** property. The following screenshot shows the WebPlanner rendered with **TimeUnit** set to 15.

Dr. Robinson	
7:00 AM	
7:15 AM	
7:30 AM	
7:45 AM	
8:00 AM	
8:15 AM	
8:30 AM	
8:45 AM	
9:00 AM	

The DayController component puts the WebPlanner in Day mode. To configure the DayController, do the following:

1. Set the **StartTime** property to the initial time to be displayed in the sidebar. The StartTime is of type TimeSpan. To specify the time, enter the hour, in 24 hour format, followed by a colon and the minutes. For example, 7:00 AM would be entered as 7:00 and 5:00 PM would be entered as 17:00.
2. Set the **EndTime** property to the final time to be displayed in the sidebar.
3. Set the **ActiveStartTime** property to the beginning of the active time zone.
4. Set the **ActiveEndTime** property to the end of the active time zone.
5. Specify the current date for the WebPlanner via the **Date** property.
6. Set the number of days to display via the **NumberOfDays** property.
7. Set the **DayIncrement** property to the number of days the view will shift when the DayController's Next and Prev methods are called. The default value of 1 is suitable for most situations.
8. Specify how PlannerEvents are to be organized via the **DayMapping** property.

In Day mode, the PlannerEvents are organized according to the value of the **DayMapping** property. The DayMapping property is found in the WebPlanner's **Mode** property.

MultiDay DayMapping

MultiDay displays one position per day for the total number of days specified via the DayController or the WebPlanner's **Mode** property. The starting date is specified via the DayController's **Date** property or the CurrentDate property of the WebPlanner's **Mode**. The events displayed for each day may be associated with any of the [resources](#) contained in the **Resources** collection. The following screenshot shows a WebPlanner in Day mode, configured to show 4 consecutive days. The WebPlanner automatically fills the **Caption** property of each position with the correct date string.

	7/16/2004	7/17/2004	7/18/2004	7/19/2004
7:00 AM				

MultiResource DayMapping

MultiResource displays one position per Resource in the WebPlanner's **Resources** collection. Use the DayController's **Date** property to specify the date of the events retrieved for the resources. The WebPlanner automatically fills the **Caption** property of each position with the name of the corresponding resource.

	Dr. Robinson	Dr. Atkinson	Dr. Johnson	Dr. Simpson
7:00 AM				

MultiDayResource DayMapping

MultiDayResource combines the MultiDay and MultiResource mappings. The WebPlanner displays the resources grouped by day. Set the value of the DayController's **NumberOfDays** property to the number of days to be displayed. This is also equivalent to the number of groups displayed. For each day that is displayed, the WebPlanner renders one subcolumn per Resource in the WebPlanner's **Resources** collection. The initial date displayed is governed by the DayController's **Date** property. The WebPlanner automatically fills the group captions with the dates and the subcolumn captions with the names of the resources. In the following screenshot, the **Resources** collection contains 3 resources, the starting date is 7/16/2004, and the number of days is 2.

	Dr. Robinson		Dr. Atkinson		Dr. Johnson	
	7/16/2004	7/17/2004	7/16/2004	7/17/2004	7/16/2004	7/17/2004
7:00 AM						

MultiResourceDay DayMapping





MultiResourceDay is another mapping that combines the MultiDay and MultiResource mappings. The WebPlanner displays one group per resource in the **Resources** collection. Each group contains one subcolumn per day. The number of days displayed per group is governed by the DayController's **NumberOfDays** property. Use the DayController's **Date** property to specify the starting date. The WebPlanner automatically fills the group captions with

the names of the resources and the subcolumns with the date of the corresponding day. In the following screenshot, the Resources collection contains 3 resources, the starting date is 7/16/2004, and the number of days displayed is 2.

	7/16/2004			7/17/2004		
	Dr. Robinson	Dr. Atkinson	Dr. Johnson	Dr. Robinson	Dr. Atkinson	Dr. Johnson
7:00 AM						

## Day Period

Use Day Period mode to display a range of days. When in Day Period mode, each cell of the time axis represents an entire day. Cells are displayed as active or inactive based upon the WebPlanner's **InactiveDays** property. If a day is active and the WebPlanner's **Bands** are active, the day is drawn using the **ActivePrimary** and **ActiveSecondary** colors. If a day is inactive, it is rendered using the **InactivePrimary** and **InactiveSecondary** colors. The following screenshot shows an example of Day Period mode.





	Carlson Caverns	Beneath the Planet of the Apes
Wed 9/10/2003		
Thu 9/11/2003		Cornelius  
Fri 9/12/2003	OMG Tours  	
Sat 9/13/2003		

The WebPlanner renders one column per position in the **Positions** collection. The position is mapped to the corresponding resource in the **Resources** collection. The DayPeriodController component puts the WebPlanner into Day Period mode. To configure the DayPeriodController, do the following:

1. Set the **StartDate** property to the initial date to be displayed in the WebPlanner.
2. Set the **EndDate** property to the final date to be displayed.
3. As an alternative to setting **EndDate**, specify a value for the **NumberOfDays** property. The DayPeriodController will automatically calculate the **EndDate**.
4. Set the **DayIncrement** to the number of days the Date is to be incremented when the controller's **Next** and **Prev** methods are called.

## Half Day Period

Use Half Day Period to schedule items on a morning or afternoon/evening basis. Each cell of the time axis represents the first or second half of a day (i.e., AM or PM). Cells are displayed as active or inactive based upon the WebPlanner's **InactiveDays** property. If a day is active and the WebPlanner's **Bands** are active, the day is drawn using the **ActivePrimary** and **ActiveSecondary** colors. If a day is inactive, it is rendered using the **InactivePrimary** and **InactiveSecondary** colors. The following screenshot shows an example of Day Period mode. The following screenshot shows an example of Half Day Period mode.

	Carlson Caverns	Beneath the Planet of the Apes
Wed 9/10/2003 AM		
Wed 9/10/2003 PM		
Thu 9/11/2003 AM		Cornelius  
Thu 9/11/2003 PM		
Fri 9/12/2003 AM	OMG Tours  	
Fri 9/12/2003 PM		
Sat 9/13/2003 AM		
Sat 9/13/2003 PM		

The WebPlanner renders one column per position in the **Positions** collection. The position is mapped to the corresponding resource in the **Resources** collection. The HalfDayPeriodController component puts the WebPlanner into Day Period mode. To configure the HalfDayPeriodController, do the following:

1. Set the **StartDate** property to the initial date to be displayed in the WebPlanner.
2. Set the **EndDate** property to the final date to be displayed.
3. As an alternative to setting **EndDate**, specify a value for the **NumberOfDays** property. The controller will automatically calculate the **EndDate**.
4. Set the **DayIncrement** to the number of days the Date is to be incremented when the controller's **Next** and **Prev** methods are called.

## Month

Use Month mode to display the scheduled events for an entire month. To display information for multiple months at once, use [MultiMonth](#) mode. Each cell of the time axis represents an entire day. Cells are displayed as active or inactive based upon the WebPlanner's **InactiveDays** property. If a day is active and the WebPlanner's **Bands** are active, the day is drawn using the **ActivePrimary** and **ActiveSecondary** colors. If a day is inactive, it is rendered using the **InactivePrimary** and **InactiveSecondary** colors. The following screenshot shows an example of Month mode.

	Carlson Caverns	Beneath the Planet of the Apes
Mon 9/1/2003		
Tue 9/2/2003		
Wed 9/3/2003		
Thu 9/4/2003		
Fri 9/5/2003		
Sat 9/6/2003		
Sun 9/7/2003		

The WebPlanner renders one column per position in the **Positions** collection. The position is mapped to the corresponding resource in the **Resources** collection. The MonthController component puts the WebPlanner into Month mode. To configure the MonthController, do the following:

1. Set the **Month** property to the number of the month to be displayed. The value 1 corresponds to January, 2 is February, and so on.
2. Set the **Year** property to the number of the year to be displayed.

The MonthController's **Next** and **Prev** methods shift the Month by 1.

## MultiMonth

Use MultiMonth mode to display events scheduled across a contiguous range of months. To display items for one month at a time, use [Month](#) mode. Each cell of the time axis represents an entire day. Cells are displayed as active or inactive based upon the WebPlanner's **InactiveDays** property. If a day is active and the WebPlanner's **Bands** are active, the day is drawn using the **ActivePrimary** and **ActiveSecondary** colors. If a day is inactive, it is rendered using the **InactivePrimary** and **InactiveSecondary** colors. The WebPlanner renders one position per month displayed. Events are retrieved and rendered for all of the resources in the WebPlanner's **Resources** collection. The following screenshot shows an example of MultiMonth mode.



	September	October	November
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11	Cornelius		
12	OMG Tours		

The MultiMonthController component puts the WebPlanner into MultiMonth mode. To configure the MultiMonthController, do the following:

1. Set the **StartMonth** property to the number of the month to be displayed in the first position. The value 1 corresponds to January, 2 is February, and so on.
2. Set the **Year** property to the year of the **StartMonth**. The WebPlanner calculates the year for the subsequent months shown in the WebPlanner.
3. Set the **NumberOfMonths** property to the number of months to be displayed. In the previous screenshot, **NumberOfMonths** was set to the value 3.

## Timeline

Timeline mode is similar to [Day](#) mode, in that each cell of the time axis represents hours and minutes of the day. However, Timeline allows information to be displayed for a range of days whereas Day mode only displays information for one day. The WebPlanner's **PositionCount** controls the number of positions and each position is mapped to the corresponding resource in the **Resources** collection.

The following screenshot shows an example of a WebPlanner rendered in Timeline mode:





	Carlson Caverns	Beneath the Planet of the Apes
0:00 AM 9/17/2003		
0:30 AM		
1:00 AM		
1:30 AM		
2:00 AM		
2:30 AM		
3:00 AM		
3:30 AM		
4:00 AM		
4:30 AM		

The TimelineController component puts the WebPlanner into Timeline mode. To configure the TimelineController, do the following:

1. Set the **StartDate** to the initial date to be displayed in the WebPlanner. In the previous screenshot, the **StartDate** was set to 9/17/2003.
2. Set the **EndDate** to the final date to be displayed. Alternatively, use the **NumberOfDays** property to indicate the number of days to be rendered. The WebPlanner will then calculate the correct value for the **EndDate** property.
3. Set the **DayIncrement** to the number of days the StartDate is to be adjusted when the controller's **Next** and **Prev** methods are called. The default value of 1 should be suitable for most situations. Note that both the **StartDate** and **EndDate** are adjusted when **Next** and **Prev** are called.

## Week

Use Week mode to display scheduled events for a range of weeks. The first date displayed is the first day of the first week indicated by the WeekController's **Month** property. The start of the week (i.e., is Sunday or Monday the first day of the week?) is controlled via the **WeekStart** property. Each cell of the time axis represents an entire day. Cells are displayed as active or inactive based upon the WebPlanner's **InactiveDays** property. If a day is active and the WebPlanner's **Bands** are active, the day is drawn using the **ActivePrimary** and **ActiveSecondary** colors. If a day is inactive, it is rendered using the **InactivePrimary** and **InactiveSecondary** colors. The number of positions rendered is equal to the WebPlanner's **PositionCount**. The WebPlanner maps each position to the corresponding resource in the **Resources** collection. The following screenshot shows an example of Week mode.

	Carlson Caverns	Beneath the Planet of the Apes
Sun 8/31/2003		
Mon 9/1/2003		
Tue 9/2/2003		
Wed 9/3/2003		
Thu 9/4/2003		
Fri 9/5/2003		
Sat 9/6/2003		
Sun 9/7/2003		
Mon 9/8/2003		
Tue 9/9/2003		
Wed 9/10/2003		
Thu 9/11/2003		Cornelius  
Fri 9/12/2003	OMG Tours  	

The WeekController component puts the WebPlanner into Week mode. To configure the WeekController, do the following:

1. Set the **Month** property to the number of the initial month to be displayed. The value 1 corresponds to January, 2 to February, and so on.
2. Set the **Year** property to the number of the year to be displayed.
3. Set the **Weeks** property to the number of weeks to be displayed.

The controller's **Next** and **Prev** methods shift the month by 1.

## ActiveDay

Use the ActiveDayController to display a series of multiple days where only active days are shown. By default, inactive days are Saturday and Sunday but this can be customized with the property WebPlanner.InActiveDays. As such, when NumberOfDays in ActiveDayController is set to 10 and Saturday, Sunday are marked as inactive days, the WebPlanner will show 2 consecutive series of days Monday to Friday.

## DisjunctDay

Use the DisjunctDayController to have the WebPlanner show a non consecutive series of days. The days that the WebPlanner should display are setup using the DisjunctDayController.Dates collection. This is a strongly-typed collection of DateTime's holding the dates for display.

To setup the DisjunctDayController with dates to view, code similar to the snippet below can be used:

```
DisjunctDayController1.Dates.Clear();
```

```
DisjunctDayController1.Dates.Add(new TMS.WebPlanner.DateItem(new DateTime(2005,10,25)));  
DisjunctDayController1.Dates.Add(new TMS.WebPlanner.DateItem(new DateTime(2005,10,28)));  
DisjunctDayController1.Dates.Add(new TMS.WebPlanner.DateItem(new DateTime(2005,10,31)));
```

this will add the days October 25, October 28 and October 31 in the WebPlanner.





# MonthPlanner

The MonthPlanner is similar in scope and functionality to the [WebPlanner control](#), except that it looks and feels like a wall calendar. Use the MonthPlanner to display [events](#) (e.g., meetings, appointments, classes) within any month of a given year. Events may span one or more days. The events may be programmatically added to the MonthPlanner or retrieved from a database via a [DataStore](#) component. The user may create, edit, delete, and resize events through the MonthPlanner interface. Events may be moved via drag and drop.

One difference between the MonthPlanner and WebPlanner is that the MonthPlanner pays no attention to the [resource](#) with which an event is associated. If an event is associated with a resource (e.g., class room, piece of equipment, person) then that relationship is retained. However, that kind of relationship does not influence how or where the event is rendered within the MonthPlanner control.

## Geography

The MonthPlanner is comprised of several sections, as shown in the following screenshot.

January 2005						
Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	31	1 +
2	+3	+4	+5	+6	+7	+8 +
Sales Bring your own lunch.		New Bring your best stories.	Donuts donuts donuts  			
				Accounts  		
9	+10	+11	+12	+13 Goal meeting	+14	+15 +
16	+17	+18	+19	+20	+21	+22 +
23	+24	+25	+26	+27	+28	+29 +
30	+31	+1	2	3	4	5

### 1: Title

The Title area identifies the month and year for which events are being displayed. It also provides controls for moving to the previous or next month or year. Use the MonthPlanner's [TitleStyle](#) property to configure the title.

## 2: Day header

There are 7 columns in the MonthPlanner, one for each day of the week. The Day header identifies the day associated with each column. Use the MonthPlanner's **DayHeaderStyle** property to control the appearance of the header. See the [Day header](#) section for more information.

## 3: Other month days

The MonthPlanner always displays 6 rows of days. The first and last rows may display one or more days from the previous and following months. These areas will display events from those months and the user is allowed to drag existing events into those areas. Use the [OtherMonthDayStyle](#) property to set the color and font information for this area.

## 4: Current month days

The largest portion of the MonthPlanner is devoted to showing events (e.g., appointments, meetings) for all the days of the current month. Any given event may be limited to a single day or stretch across multiple days. The appearance of the days shown in this is controlled via three MonthPlanner properties: **DayStyle**, **InactiveDayStyle**, and **TodayStyle**. These styles are described in the [Current month days](#) section.

## Displaying Data

Use the MonthPlanner's **Month** and **Year** properties to control the specific month and year for which events (e.g., meetings, appointments) are to be displayed. Only those events contained in the MonthPlanner's **Items** collection are considered for display. Events are represented by instances of the [PlannerEvent](#) class and the Items collection may contain zero or more PlannerEvents. Not every event in that collection will be displayed. An event is displayed only if it is within the specified month and year, or within one of the [Other Month](#) days visible in the MonthPlanner.

PlannerEvents are loaded into the Items collection in one of two ways:

- The application programmatically creates instances of the PlannerEvent class and adds them to the Items collection. For more information about this approach, see the section titled [Managed PlannerEvents by hand](#).
- The application connects the MonthPlanner to a [DataStore](#) and the MonthPlanner retrieves the events through the DataStore.

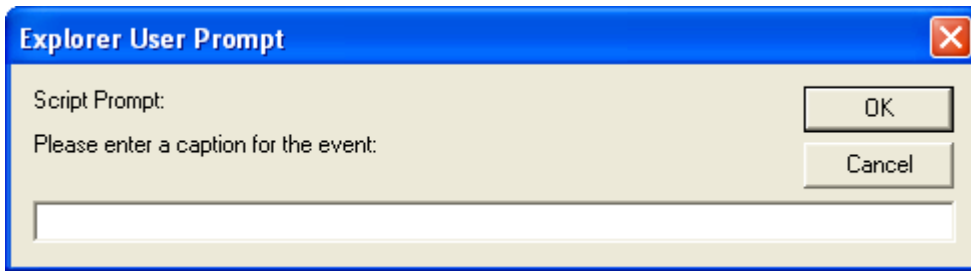
An application may choose to use a combination of both methods. For example, if certain time periods may not be scheduled then the application may create an instance of PlannerEvent for each time period, add it to the Items collection, and set the event's **Background** property to the value True. It could then let the MonthPlanner load scheduled events from Microsoft SQL Server through a SqlClientDataStore.

## Create and Edit Events

Users can create new events in any of the following ways:

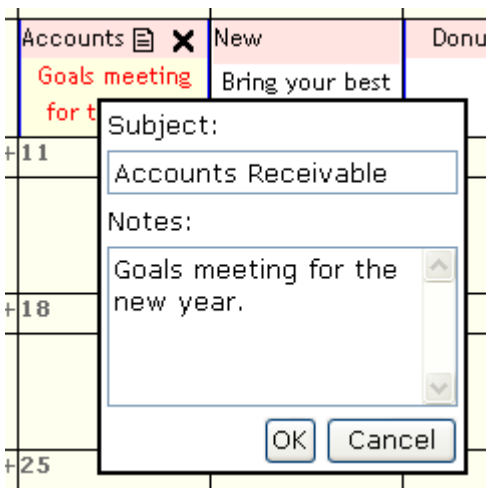
- Clicking the Insert glyph (i.e., '+') in the header of a day cell.
- Clicking on a day cell.
- Selecting a contiguous range of day cells.

The default behavior of the MonthPlanner is to display a client-side dialog built into the browser. The dialog asks the user to enter the caption of the event.



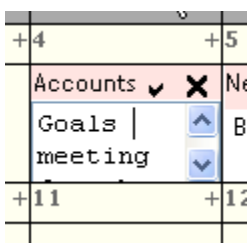
Once the user has created the event, they may then enter its notes by editing the event. The MonthPlanner has two built-in edit modes: Popup editing and In Place editing. Two custom modes are also available but they require you to perform additional work. For more information about those modes, see the [Custom Editing](#) section.

To change the editing mode, use the MonthPlanner's **EditType** property. To place the planner in Popup Edit mode, set its value to **PopupEdit**. In this mode, the user edits an event by clicking on the edit glyph displayed in the event's caption, by clicking the selected event once, or by clicking an event twice (once to select it, once to edit it). In all three cases, the MonthPlanner displays a popup edit window.



Clicking the OK button saves the changes to the event. Clicking cancel closes the edit window. Use the **MonthPlanner.PopupEdit** composite property to control the colors, font, and labels displayed in the popup edit window.

If the MonthPlanner's EditType is set to **InPlace**, the event notes can be edited by clicking in the body of the event. The body turns into an edit window with a vertical scroll bar. The text can be changed in place. To save the changes, click the checkmark in the event's caption. To cancel the changes, click the "X".



When in InPlace editing mode, right click on the caption in order to change it. The MonthPlanner displays the same edit dialog shown at the beginning of this section. The default dialogs and editing behaviors may not be suitable for some applications. The MonthPlanner has two custom edit modes that allow you to insert your own ASP.NET web forms in place of MonthPlanner's dialogs and edit windows. For more information on this topic, please read the following section.

## Custom Editing

The dialogs used by the MonthPlanner for creating and editing events are very basic. They have a simple look and allow input of an event's caption and notes only. Many applications will find the need to collect additional information and/or enforce certain business rules related to the scheduling of events. The MonthPlanner provides a lot of flexibility in this area. Using the EditType property, an application can place the planner into one of two custom editing modes. The modes are described in the following sections.

### Detail Edit Mode

Use Detail Edit mode when you want your user to create or edit events using your own ASP.NET web form instead of the default popup. For example, the user may need to enter application-specific information in addition to the meeting's topic or notes. Do not use this mode if other types of actions are to be performed when the user selects cells. Instead, use [Custom mode](#).

Two types of detail editing are possible. When EditType is set to DetailEdit, a popup window showing the form to perform the editing is used. When EditType is set to DetailEditEmbedded, the form for editing is shown inside a dialog inside the same browser window as the WebPlanner. The visual appearance of this dialog can be further customized with the properties WebPlanner.DetailEdit.EmbeddedBackColor, WebPlanner.DetailEdit.EmbeddedCaptionColor, WebPlanner.DetailEdit.EmbeddedCaptionForeColor.

**Note:** To see an excellent example of this mode, look at the Meeting Rooms portion of the example application installed with WebPlanner. The example application is installed in the <WebPlanner root>\Examples directory.

To place the MonthPlanner in Detail Edit mode, set its **EditType** property to the value **DetailEdit**. When the user attempts to create a new event or edit an existing event, the planner opens the ASP.NET web form identified via the **MonthPlanner.DetailEdit.EditTarget** property. If the user attempts to view a read-only event, the planner opens the ASP.NET web form identified via the **MonthPlanner.DetailEdit.ViewTarget** property. For this mode to function correctly, those properties must be filled with the relative or absolute URL of a web form.

Another requirement for Detail Edit mode is that the MonthPlanner's **SelectionMode** property must be set to the value **SingleSelectAutoCreate** or **MultiSelectAutoCreate**.

The custom edit form is responsible for carrying out any required actions, saving or updating data in the application's database, and notifying the planner when the edit form is closing. The edit form can determine what it needs to do by looking at the parameters passed to it in the HttpRequest. When the form is invoked, the planner places several parameters in the HttpRequest. They provide enough information for the form to determine where a new event is to be placed or what event is being edited. The following table identifies the parameters:

Parameter name	Value
DetailType	Identifies the type of action being performed by the user. Values are "Create", "Edit", and "View".
EventKey	When editing or viewing a PlannerEvent, the value of this parameter is the KeyID (i.e., unique ID) of the event.
SelEnd	When adding a new event, the ending cell selected by the user. This is a base zero column number.
SelPos	The base zero row number selected by the user.
SelStart	When adding a new event, the beginning cell selected by the user. This is a base zero column number.

So that every developer does not have to spend time writing code to retrieve these parameters, WebPlanner includes a utility class that retrieves the parameters for you. The class is named **DetailUtil** and it has a method named **GetParameters**. If the application passes the page's HttpRequest (i.e., Page.Request) to the GetParameters method, that method will return an instance of class **DetailEditParameters**. The DetailEditParameters class has a property for each of the aforementioned parameters.



When creating a new event, the custom edit form needs to translate the SelEnd, SelPos, and SelStart parameters into meaningful DateTime values. The MonthPlannerUtil class provides a **CellToDateTime** method that converts a row number and column number to a DateTime value.

When the custom edit form is ready to close, it should cause the invoking MonthPlanner to refresh itself. The refresh forces the MonthPlanner to reload its events and the new event or changes to an existing event will appear. To cause the refresh, use the **DetailUtil.RefreshParentWindow** method. This method registers a client-side script that tells the edit form's parent window to refresh itself and then closes the edit form.

The following code examples show how a custom edit form obtains the Detail Edit parameters:

### C# Example

```
private void InitDetails()
{
    ...
    // Grab the detail edit parameters out of the request.
    DetailEditParameters parameters = DetailUtil.GetParameters(Request);
    this.SelEnd = parameters.SelectionEnd;
    this.SelPos = parameters.SelectionPosition;
    this.SelStart = parameters.SelectionStart;

    Meeting meeting = null;
    if (parameters.EventKey > 0)
    {
        meeting = new Meeting(parameters.EventKey);
        this.Meeting = meeting;
    }
    ...
}
```

### VB.NET Example

```
Private Sub InitDetails()
    ...
    ' Grab the detail edit parameters out of the request.
    Dim parameters As DetailEditParameters = DetailUtil.GetParameters(Request)
    Me.SelEnd = parameters.SelectionEnd
    Me.SelPos = parameters.SelectionPosition
    Me.SelStart = parameters.SelectionStart

    Dim meeting As Meeting = Nothing
    If parameters.EventKey > 0 Then
        meeting = New Meeting(parameters.EventKey)
        Me.Meeting = meeting
    End If
    ...
End Sub
```

The following code example shows how the Save and Cancel buttons on a custom edit form use the MonthPlannerUtil.CellDateToTime and DetailUtil.RefreshParentWindow methods. Note that this source code is taken from the Meeting Rooms example mentioned earlier in this section.

### C# Example

```
private void buttonSave_Click(object sender, System.EventArgs e)
{
```

```

// Perform actions required to save the event.
Meeting meeting = this.Meeting;
if (meeting == null)
{
    #region Add

    // Get the planner.
    PlannerBucket bucket = this.GetPlanner();

    // Create the meeting
    meeting = new Meeting();
    meeting.Topic = textBoxTopic.Text;
    meeting.Notes = textBoxNotes.Text;
    meeting.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue);
    meeting.CreatorID = this.CreatorID;

    if (bucket.WebPlanner != null)
    {
        meeting.StartTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner,
            this.SelStart, this.SelPos, true);
        meeting.EndTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner,
            this.SelEnd, this.SelPos, false);
    }
    else
    {
        meeting.StartTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner,
            this.SelPos, this.SelStart);
        meeting.EndTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner,
            this.SelPos, this.SelEnd);
    }
    meeting.Insert();

    #endregion Add
}
else
{
    #region Edit

    meeting.Topic = textBoxTopic.Text;
    meeting.Notes = textBoxNotes.Text;
    meeting.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue);
    meeting.Update();

    #endregion Edit
}

// Register a start up script that refreshes the
// main window & closes this window.
DetailUtil.RefreshParentWindow(Page, true);
}

private void buttonCancel_Click(object sender, System.EventArgs e)
{
    // Force parent page to refresh. In the case of adding a new
    // meeting, this will deselect the selected cells.
    DetailUtil.RefreshParentWindow(Page, true);
}

```

```

Private Sub buttonSave_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles buttonSave.Click

    ' Perform actions required to save the event.
    Dim mtg As Meeting = Me.Meeting
    If (Meeting Is Nothing) Then
        ' Get the planner.
        Dim bucket As PlannerBucket = Me.GetPlanner()

        ' Create the meeting
        mtg = New Meeting
        mtg.Topic = textBoxTopic.Text
        mtg.Notes = textBoxNotes.Text
        mtg.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue)
        mtg.CreatorID = Me.CreatorID

        If (Not bucket.WebPlanner Is Nothing) Then
            mtg.StartTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner, _
                Me.SelStart, Me.SelPos, True)
            mtg.EndTime = WebPlannerUtil.CellToDateTime(bucket.WebPlanner, _
                Me.SelEnd, Me.SelPos, False)
        Else
            mtg.StartTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner, _
                Me.SelPos, Me.SelStart)
            mtg.EndTime = MonthPlannerUtil.CellToDate(bucket.MonthPlanner, _
                Me.SelPos, Me.SelEnd)
        End If
        mtg.Insert()

    Else
        mtg.Topic = textBoxTopic.Text
        mtg.Notes = textBoxNotes.Text
        mtg.RoomID = Convert.ToInt32(dropDownRooms.SelectedValue)
        mtg.Update()
    End If

    ' Register a start up script that refreshes the
    ' main window & closes this window.
    DetailUtil.RefreshParentWindow(Page, True)

End Sub

Private Sub buttonCancel_Click(ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles buttonCancel.Click

    ' Force parent page to refresh. In the case of adding a new
    ' meeting, this will deselect the selected cells.
    DetailUtil.RefreshParentWindow(Page, True)

End Sub

```

## Custom Mode

Custom mode is very similar to [Detail Edit mode](#). The main difference is that it does not automatically invoke the custom edit form specified in the MonthPlanner's **DetailEdit** property. Instead, the application must invoke the form programmatically. An example of doing so is provided at the end of this section.

To put the MonthPlanner into custom mode, do the following:

- Set the MonthPlanner's **EditType** property to **Custom**.
- Set the MonthPlanner's **SelectionMode** property to **SingleSelect** or **MultiSelect**.

In custom mode, the application knows that the user wishes to create an event when the MonthPlanner's **CellsSelected** event is raised. If the SelectionMode is SingleSelect, the CellsSelected event is raised when the user has clicked on a single cell in the MonthPlanner. If the SelectionMode is MultiSelect, the event is raised after the user selects one or more cells.

The application knows that the user wants to edit an event when the MonthPlanner's **EventCustomEdit** event is raised. MonthPlanner raises EventCustomEdit when the user clicks on an edit glyph in the event's caption, when the user clicks the currently selected event, or when the user double clicks an event (the first click selects, the second raises the event).

The application is responsible for passing the appropriate parameters to and invoking the custom edit form. The custom edit form must then retrieve the parameters, configure its interface, and save any changes made by the user to the database. When the user has saved or canceled their changes, the edit form must refresh its parent window and close itself using the DetailUtil.RefreshParentWindow method. Doing so causes the MonthPlanner to refresh and reload the events for the currently displayed month.

The following code example is from an existing application and shows how can invoke its custom edit form. Note that some of the code, such as the CustomPrincipal class, is specific to the application.

### C# Example

```
// Called when the user is creating an event.
private void webPlanner_CellsSelected(object sender,
    TMS.WebPlanner.PlannerCellsSelectedEventArgs e)
{
    // Create an event for the selected cells.
    PlannerEvent plannerEvent = webPlanner.CreateEventAtSelection();

    // Add information about the meeting to the session. It will be
    // picked up by the popup window.

    Session.Add("PlannerEvent", plannerEvent);
    Session.Add("RoomID", e.SelectedResource.ResourceID);

    CustomPrincipal principal = (CustomPrincipal) Page.User;
    if (principal != null)
        Session.Add("CreatorID", principal.Person.ID);
    else
        throw new CustomException("No principal found when creating a new event.");

    // Add startup script that displays a popup to capture the meeting details.
    string script = "<script language='javascript'> " +
        "window.open('DetailPopup.aspx', 'AddMeeting'," +
        "'width=400, height=350, menubar=no, center=yes, resizable=no');" +
        "</script>";
    Page.RegisterStartupScript("AddMeeting", script);
}

// Called when the user is editing an event.
private void webPlanner_EventCustomEdit(object sender,
    TMS.WebPlanner.PlannerEventClickEventArgs e)
{
    // Add information about the meeting to the session. It will be
```

```
// picked up by the popup window.
Session.Add("RoomID", e.Event.ResourceID);
Session.Add("MeetingID", e.Event.Key);
CustomPrincipal principal = (CustomPrincipal) Page.User;
if (principal != null)
    Session.Add("CreatorID", principal.Person.ID);
else
    throw new CustomException("No principal found when creating a new event.");

// Add startup script that displays a popup to capture the meeting details.
string script = "<script language='javascript'> " +
    "window.open('DetailPopup.aspx', 'EditMeeting'," +
    "'width=400, height=350, menubar=no, center=yes, resizable=no');" +
    "</script>";
Page.RegisterStartupScript("AddMeeting", script);
}
```

## VB.NET Example

```
' Called when the user is creating an event.
Private Sub webPlanner_CellsSelected(sender As Object, _
    e As TMS.WebPlanner.PlannerCellsSelectedEventArgs)

    ' Create an event for the selected cells.
    Dim plannerEvent As PlannerEvent = webPlanner.CreateEventAtSelection()

    ' Add information about the meeting to the session. It will be
    ' picked up by the popup window.
    Session.Add("PlannerEvent", plannerEvent)
    Session.Add("RoomID", e.SelectedResource.ResourceID)

    Dim principal As CustomPrincipal = CType(Page.User, CustomPrincipal)
    If Not (principal Is Nothing) Then
        Session.Add("CreatorID", principal.Person.ID)
    Else
        Throw New CustomException("No principal found when creating a new event.")
    End If
    ' Add startup script that displays a popup to capture the meeting details.
    Dim script As String = "<script language='javascript'> " + _
        "window.open('DetailPopup.aspx', 'AddMeeting'," + _
        "'width=400, height=350, menubar=no, center=yes, resizable=no');" + _
        "</script>"
    Page.RegisterStartupScript("AddMeeting", script)

End Sub

' Called when the user is editing an event.
Private Sub webPlanner_EventCustomEdit(sender As Object, _
    e As TMS.WebPlanner.PlannerEventClickEventArgs)

    ' Add information about the meeting to the session. It will be
    ' picked up by the popup window.
    Session.Add("RoomID", e.Event.ResourceID)
    Session.Add("MeetingID", e.Event.Key)
    Dim principal As CustomPrincipal = CType(Page.User, CustomPrincipal)
    If Not (principal Is Nothing) Then
        Session.Add("CreatorID", principal.Person.ID)
    Else
        Throw New CustomException("No principal found when creating a new event.")
    End If
End Sub
```

```
' Add startup script that displays a popup to capture the meeting details.
Dim script As String = "<script language='javascript'> " + _
    "window.open('DetailPopup.aspx', 'EditMeeting'," + _
    "'width=400, height=350, menubar=no, center=yes, resizable=no');" + _
    "</script>"
Page.RegisterStartupScript("AddMeeting", script)
```

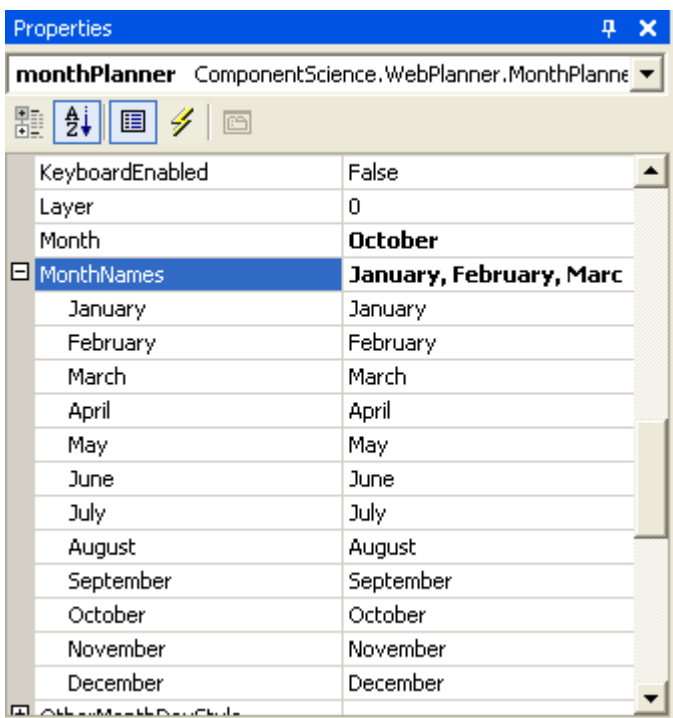
End Sub

## Title

The Title area of the MonthPlanner shows the month and year for which PlannerEvents are being displayed. It also provides controls that allow the user to shift the view to the previous or next month or year. For example, clicking the '<' glyph moves to the previous month. Clicking the '<<' glyph moves to the same month of the previous year. The '>' and '>>' glyphs move to the following month and the same month of the following year, respectively.

By default, the name of the month and the year are displayed in the title area. The month names default to the names used by the current culture on the computer hosting the web application. To customize the names, change the values within the **MonthNames** property. To hide the year, set the MonthPlanner's **ShowYearInTitle** property to the value **false**.

The following screenshot shows the initial month names in the Visual Studio.NET Property grid for the EN-US culture.



## Custom Glyphs

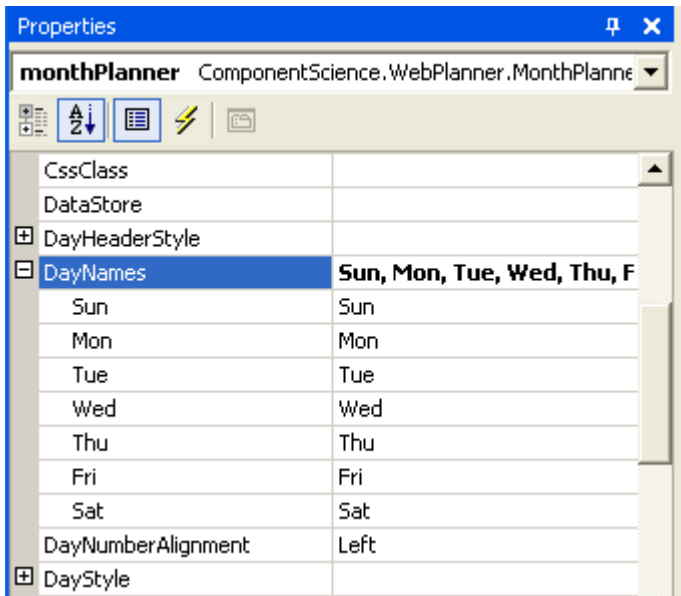
To use different glyphs for shifting between months and years, specify the URLs of the glyphs in the MonthPlanner's **Glyphs** properties. The Glyphs property is a composite property containing a slot for each custom glyph. For example, the MonthPlanner.Glyphs.NextMonth property identifies the image to be displayed in place of '>'. Note that the value of these properties should be an absolute or relative URL. For example, "http://mydomain.com/images/NextMonth.gif" or "../images/NextMonth.gif". Do not specify file paths for these properties. It will work at design time because the web server is able to locate the files on your local hard drive. However, it most likely will not work in a production environment.

Title Style

To control the colors and fonts used to render the Title area, use the MonthPlanner's **TitleStyle** composite property. By default, the Title is rendered in a solid color. To create a gradient effect, provide two different colors for the **TitleStyle.BackColor** and **BackColorTo** properties. To control the height of the title, use the **TitleStyle.Height** property. Out of the box, the title text is rendered in black. To change this, use the **TitleStyle.ForeColor** property. The **TitleStyle.Font** property controls the font used for the text, the font's size, and whether it is bolded.

## Day Header

The purpose of the day header is to identify the day of the week for each column of the MonthPlanner. The day names default to those of the current culture on the computer hosting the web application. The default names can be changed via the MonthPlanner's **DayNames** property. The following screenshot shows this property in Visual Studio.NET's Property Grid.



By default, the day header considers Sunday to be the start of the week. To have the week start with a different day, set the **StartDay** property to the appropriate day of the week. The following screenshot shows the MonthPlanner with **StartDay** set to the value **DayOfWeek.Monday**.

January 2005						
Mon	Tue	Wed	Thu	Fri	Sat	Sun
27	28	29	30	31	1	+2

The appearance of the day header is controlled by the **DayHeaderStyle** composite property. Use the **DayHeaderStyle.BackColor** and **ForeColor** properties to control the background and text colors of the day header. Use the **DayHeaderStyle.Font** style to control the font attributes such as the font used and its size.

## Current Month Days

The MonthPlanner shows the days for the month specified via its **Month** and **Year** properties. It also shows a certain number of days from the preceding and following months. For more information on controlling the appearance of days from those months, see the [Other month days](#) section. The days of the current month are placed into three categories:

- Inactive days: Days of the week on which no work is to be performed (i.e., Saturday and Sunday).
- Active days: Days of the week on which work is to be performed (i.e., Monday through Friday).

- Today: The current day of the year (i.e., the value of DateTime.Now.Date).

The inactive days default to Saturday and Sunday. To change the inactive days, use the MonthPlanner's **InactiveDays** property. To change the coloring or font for the inactive days, use the **InactiveDaysStyle** property.

The color of the active days defaults to the MonthPlanner's **BackColor** and **BackColorTo** properties. To display those days with a solid background color, set the value of BackColorTo to String.Empty or give it the same value as the BackColor property. To have a gradient effect, set the properties to two different colors. Use the GradientDirection property to control whether the gradient runs from left to right or top to bottom. The following screenshot shows the MonthPlanner with a gradient.

January 2005									
<<	<	Sun	Mon	Tue	Wed	Thu	Fri	>	>>
		26	27	28	29	30	31	1	+
		2	+3	+4	+5	+6	+7	+8	+
		9	+10	+11	+12	+13	+14	+15	+
		16	+17	+18	+19	+20	+21	+22	+
		23	+24	+25	+26	+27	+28	+29	+
		30	+31	+1	2	3	4	5	

The MonthPlanner can also display a background image in place of the background colors. Use the **BackgroundImage** property to specify the image to be displayed. Use a relative or absolute URL to identify the image. Do not specify a path that is specific to your development computer.

To control the background and foreground colors of each day category, use the following properties:

- **DayStyle**: Controls the font and coloring of active days.
- **InactiveDayStyle**: Controls the font and colors of inactive days.
- **TodayStyle**: Controls the font and coloring of today.

If your application needs to control coloring for a range of days within the month, use the [Period Styles](#) feature.



By default, the MonthPlanner displays the day number and insert glyph within the header of each day. The day number identifies the day of the month. By default, the day number is displayed on the left hand side of a cell's header. To switch it to the right side or center, use the **DayNumberAlignment** property.

The insert glyph allows the user to insert an event (e.g., meeting, appointment) into the day cell. By default, the '+' character is used for this purpose. To change the glyph, specify an image URL in the **MonthPlanner.Glyphs.Insert** property.

## Other Month Days

The MonthPlanner displays up to six rows of days for the month specified via its **Month** and **Year** properties. Depending upon the month shown, the first and last row may show days from the preceding and following months. These are referred to as "other month days". When using a [DataStore](#), the MonthPlanner will retrieve events for those days. By default, the MonthPlanner will also display the day number for the other month days. To turn off the display of other month day numbers, set the value of the **ShowDaysInOtherMonth** property to **false**. To control the appearance of those days, use the **OtherMonthDayStyle** property.

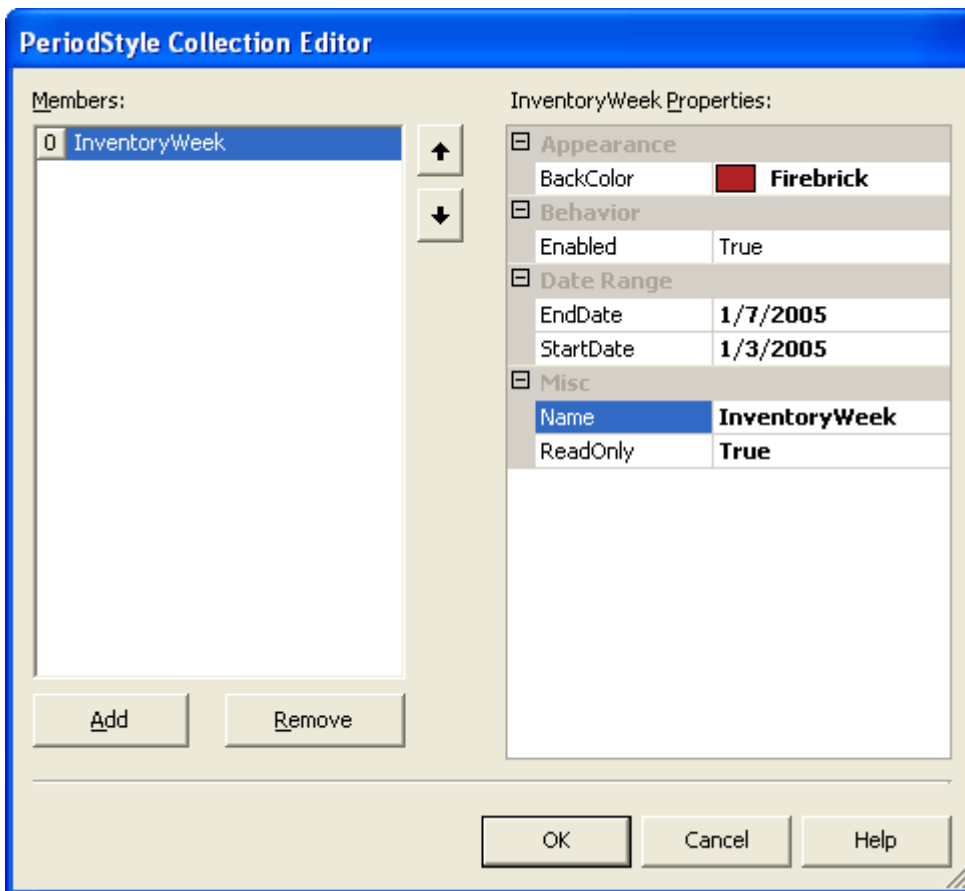
## Period Styles

An application may need to communicate special information to the user by coloring certain days of the month. Or the application may need to prevent the user from adding or changing events in certain days. *Period styles* allow an application to take care of both situations. A period style sets the background color and/or makes read-only a contiguous range of days. An application may use as many period styles as needed. If two or more period styles overlap, the first period style in the **PeriodStyles** collection always takes precedence.

To set up a period style at design time, do the following:

1. In the IDE, click the MonthPlanner and go to the Properties Window.
2. Go to the PeriodStyles property. This is a collection property.
3. Click the ellipsis button in the value window of that property. The Period Styles collection editor displays.
4. Click the Add button to create a new period style.
5. Set the **StartDate** and **EndDate** to the days covered by the style.
6. Set the **BackColor** property. If no value is specified for this property then the MonthPlanner's style properties are used for the period.
7. Set the Enabled property to True.
8. If the period is to be read only, set the **ReadOnly** property to True.
9. Click the OK button.

The following screenshot shows a period style that affects January 3rd, 2005 through January 7th, 2005. The background of the affected days is set to Firebrick and the days are read only.



Period styles may also be created at run time. The following code examples create the period style shown in the previous screenshot.

### C# Example

```
using TMS.WebPlanner;
...
private void CreatePeriodStyle()
{
    PeriodStyle style = new PeriodStyle();
    style.BackColor = Color.Firebrick;
    style.ReadOnly = true;
    style.StartDate = new DateTime(2005, 1, 3);
    style.EndDate = new DateTime(2005, 1, 7);
    monthPlanner.PeriodStyles.Add(style);
}
```

### VB.NET Example

```
Imports TMS.WebPlanner
...
Private Sub CreatePeriodStyle()
    Dim style As New PeriodStyle
    style.BackColor = Color.Firebrick
    style.ReadOnly = True
    style.StartDate = New DateTime(2005, 1, 3)
    style.EndDate = New DateTime(2005, 1, 7)
    MonthPlanner1.PeriodStyles.Add(style)
```

End Sub

## Default Event

The **DefaultEvent** property defines a PlannerEvent that is used as the model for all PlannerEvents instantiated via the [Items](#) collection editor and the [DataStores](#). For example, if you wish for all PlannerEvents to be read only then set the DefaultEvent's **ReadOnly** property to the value **true**. If your application needs to programmatically create new PlannerEvents, use the MonthPlanner's **CreateEvent** method to clone the DefaultEvent. The CreateEvent method also adds the newly-created event to the MonthPlanner's Items collection.

## Items

The Items collection is the location in which the MonthPlanner stores the PlannerEvents that are to be rendered. When the MonthPlanner is connected to a [DataStore](#) in ASP.NET 1.1 or DataSource in ASP.NET 2.0, this collection is refreshed upon each page request or postback. The logic for doing so is described in the following steps:

1. Remove all PlannerEvents whose **Origin** property is set to the value **ItemOrigin.DataStore**. This means that PlannerEvents programmatically added by the application are left in the collection. If the MonthPlanner's **PreserveApplicationItems** property is set to the value **false** then all events are removed regardless of their origin.
2. Call the DataStore's Read method to pull in the collection of events matching the current mode.
3. For each PlannerEvent retrieved by the DataStore, raise the **EventRetrieved** event. When all events have been retrieved, call the **EventsRetrieved** event.

Note that the Items collection may wind up containing events whose date and position fall outside the range displayed by the MonthPlanner. In that case, those events are ignored but remain in the collection until the next page request or postback.

When the DataStore retrieves information for an event, it clones the MonthPlanner's [DefaultEvent](#) property and modifies the clone's properties according to the retrieved information.

The application is free to modify the Items collection as necessary. More information about doing so can be found in the [Managing PlannerEvents by hand](#) section.

## Layers

In general, all PlannerEvents in the Items collection are displayed in the MonthPlanner. For most applications this is adequate. In other cases, users may wish to see a subset of the scheduled events based upon certain criteria. For example, each PlannerEvent may have a low, medium, or high priority and the user wishes to view only those with high priority. MonthPlanner supports this type of behavior through the user of *Layers*.

The MonthPlanner allows an application to set up arbitrary layers and assign PlannerEvents to those layers. For a hands on example of how this could be implemented, look at the Dr. Pepper's Office demo in the example application installed with this product. In the demo, appointments are assigned categories such as "Consultation", "Diagnostic", or "Routine Exam". Each appointment type is interpreted as a specific layer and the application allows the user to view specific types of appointments or combinations of appointment types.

Layers are established via the MonthPlanner's **Layer** property and the PlannerEvent's **Layer** property. Both properties are defined as type System.int but should be treated as bit flags. When the MonthPlanner.Layer property has the value zero, it means "no layer is in effect". When the PlannerEvent.Layer property has the value zero, it means "this event does not belong to a layer".

The first layer is represented by the value 1, the second layer by 2, the third layer by 4, the fourth layer by 8, and so on. To assign a PlannerEvent to the 3rd layer, set its Layer property to the value 4. PlannerEvents may be assigned to

more than one layer by ORing the bit values of the layers. For example, to assign a PlannerEvent to the 1st and 4th layers, set its Layer property to the value 9 (i.e., 1 + 8). The PlannerEvent's Layer property defaults to the value 0.

One of the best places to set the layer values for PlannerEvents is in the MonthPlanner's **EventsRetrieved** event. When this event is raised, all events have been retrieved from the [data store](#). The delegate may walk through the collection of events specified in the event arguments and set the layer based upon whatever criteria are important.

By default, the MonthPlanner's Layer property has the value zero. This means that the MonthPlanner will render all PlannerEvents regardless of their layer. To have the MonthPlanner show only those events assigned to the first layer, set its Layer property to 1. To show the second layer, set it to 2, and so on. To show more than one layer, OR the layer values together. For example, to show the 1st and 4th layers, set the MonthPlanner's Layer property to the value 9 (i.e., 1 + 8).

## Keyboard Support

By default, the user interacts with the MonthPlanner using only the mouse. To activate keyboard support, set the WebPlanner's **KeyboardEnabled** property to the value true. When keyboard support is enabled, the user can move from one cell to the next using the arrow keys. They can press the Insert key to create a new event and they can press the Delete key to delete the currently selected event.

## Client-side JavaScript

The MonthPlanner uses a large chunk of client-side JavaScript to implement features such as drag and drop and the resizing of events. By default, this code is included in every page returned to the client browser. To reduce the download times of pages containing a MonthPlanner control, the client-side JavaScript can be downloaded to the client once and cached. To enable this feature, do the following:

1. Set the MonthPlanner's **CachedScript** property to the value true.
2. Place a copy of the file monthplanner.js in the directory  
c:\inetpub\wwwroot\aspnet\_client\cs\_webplanner\2\_0\_0\_0.

When the page is requested, the MonthPlanner includes a script within the page's HTML. The script tells the client browser to download the monthplanner.js file. Because the monthplanner.js file is placed in the aspnet\_client directory structure, it may be shared among multiple web forms and web applications.

## Planner Events

---

Both the WebPlanner and MonthPlanner controls allow users to schedule time for specific resources or reasons. A scheduled time may be referred to as a meeting, appointment, or some other term, depending upon the type of application being built. A chunk of scheduled time is represented by an instance of the **PlannerEvent** class. A **PlannerEvent** contains basic pieces of information such as the start and end time of the event, and notes associated with the event. It also contains visual and behavioral pieces of information. For example, the color of the event when it is selected or a flag indicating whether the event may be moved from one position to another.

PlannerEvents (i.e., events) are stored in the **Items** collection of a WebPlanner or MonthPlanner (i.e., a planner). An application can fill the Items collection by either [programmatically instantiating](#) the events or retrieving them through the use of a [DataStore](#). When PlannerEvents are created through the use of a DataStore, they are clones of the planner's **DefaultEvent** property. The application then has a chance to customize the event instances via delegates for the **EventRetrieved** or **EventsRetrieved** events.

### Times

The **PlannerEvent** class has several time-related properties. The **PlannerStartTime** and **PlannerEndTime** properties identify the exact start and end times of the event. The **PlannerEvent** also has **StartTime** and **EndTime** properties which indicate the date and time used for rendering the event on the planner. It is important to note the difference. The WebPlanner control may be configured to display events on 15 minute boundaries. If an event starts at 9:16 AM and ends at 9:38 AM, when rendered on the WebPlanner it looks as though it starts at 9:15 AM and ends at 9:45 AM. The **PlannerStartTime** and **PlannerEndTime** properties let the planner retain the original start and end times of the event as they were retrieved from the database. The **StartTime** and **EndTime** properties let the WebPlanner know where to render the event.

### Captions and Notes

Each event has a caption or topic as well as some textual notes. The caption and notes may contain pure ASCII text or HTML markup. To get or set an event's caption, use the **PlannerEvent.Caption.Text** property. To access its notes, use either of the following properties:

- **PlannerEvent.Notes**: This is of type String and returns the entire set of notes in one string.
- **PlannerEvent.Text**: This is of type List<ItemCollection> and returns the notes broken up into individual lines.

When rendered, the default behavior of the event is to display the text of the caption in the caption area. However, it is also possible to display the event's time, or both its caption and time. Specify the information to be displayed via the **PlannerEvent.Caption.CaptionStyle** property.

### Behaviors

By default, a user may edit, delete, move, and resize any and all **PlannerEvents** rendered in a planner. At the planner level, it is possible to make the entire planner read only via its **ReadOnly** property. It's also possible to set read-only positions and periods via the WebPlanner's [PositionStyles](#) and the MonthPlanner's [PeriodStyles](#). However, applications requiring their users to log in may require more finer-grained control over what may be done to any particular event. For example, the application may not allow user A to modify events created by user B.

The **PlannerEvent** class has several properties that allow the application to maintain exact control over what the user may do to it. The properties can be initialized to specific values in the planner's **DefaultEvent** property. They can also be fine tuned to the situation via delegates for the **EventRetrieved** or **EventsRetrieved** events.

The following table lists the properties that control what a user may or may not do to a **PlannerEvent**.

**Property**

**Meaning**

AllowDelete	If false then the user may not delete the event.
AllowOverlap	If false then the user may not drag and drop another event such that it overlaps this event.
Background	If true then the event makes its time slot unavailable for the user to add new events. The event itself is read-only and may not be deleted, resized, or moved.
FixedPosition	If true then the event may not be moved to another column within the planner. The event may be resized or moved to another time slot within the same column.
FixedSize	If true then the event may not be resized. It may be moved to another time slot or column.
FixedTime	If true then the event may not be moved to another time slot and it may not be resized.
ReadOnly	If true then the event's caption and notes are fixed and may not be changed. Also, the event may not be deleted. However, the user may still move or resize the event.

## Colors

PlannerEvents can be configured to follow a common visual scheme or to have individualized color schemes based upon criteria established by the application. For example, read-only events could be displayed with a red background while read-write events are displayed in light beige. The planner's **DefaultEvent** defines the base color scheme used by all events. An application can modify the scheme via delegates for the planner's **EventRetrieved** or **EventsRetrieved** events.

The following properties control the colors and font of the event's caption:



- PlannerEvent.Caption.BackColor
- PlannerEvent.Caption.BackColorTo
- PlannerEvent.Caption.Font
- PlannerEvent.Caption.ForeColor
- PlannerEvent.Caption.GradientDirection

To achieve a gradient effect in the caption, set the **BackColor** and **BackColorTo** to complimentary colors. Use the **GradientDirection** property to create either a vertical or horizontal gradient.

The body of the event is rendered in one of two sets of colors, depending upon whether the event is selected. If the event is not selected, the PlannerEvent's **BackColor** and **ForeColor** properties are used. The background of the body cannot be rendered as a gradient. When the event is selected, the PlannerEvent's **SelectionColor** and **SelectionFontColor** are used. The **PlannerEvent.Font** property controls the font of the text rendered in the body. Use the **TrackBarColor** property to control the color of the resizing trackbars located at the top and bottom of the event. Note that the trackbar is visible only if the event may be resized.

## Flags

There are times where you users want to know that one event is different than the rest. It may be due to the nature of the event (e.g., a consultation versus a class) or due to some other piece of information associated with the event. In those cases, it is usually more helpful to flag the event with a special color or graphic instead of text your user must read. The PlannerEvent class has a **Flagged** property just for this purpose. When the Flagged property is set to the value true, a triangular marker is rendered in the upper right corner of the event's caption. The following screenshot shows an example:

April 2005		>		>>	
Wed	Thu	Fri	Sat		
30	31	1	+2	+	
		Chair Wrestling  			
-6	+7	+8	+9	+	

You can control the color of the flag via the `PlannerEvent`'s **FlagColor** property. The default color is `LawnGreen`. Because the flagging of an event can depend upon some application-specific piece of information associated with the event, the best place to set the flag is in the **EventsRetrieved** event of the `WebPlanner` or `MonthPlanner`. Following is a code snippet that illustrates the setting of `Flagged` and `FlagColor`.

### C# Example

```
private void WebPlanner1_EventsRetrieved(object sender,
TMS.WebPlanner.PlannerCollectionEventArgs e)
{
    // Flag meetings where food is provided.
    for (int i = 0; i < e.Events.Count; i++)
    {
        PlannerEvent item = (PlannerEvent)e.Events[i];
        bool hasFood = Convert.ToBoolean(item.OtherColumns["FoodProvided"]);
        item.Flagged = hasFood;
        item.FlagColor = Color.Red;
    }
}
```

### VB.NET Example

```
Private Sub WebPlanner1_ItemsRetrieved(ByVal sender As Object, _
    ByVal e As TMS.WebPlanner.PlannerCollectionEventArgs) _
    Handles WebPlanner1.EventsRetrieved

    ' Flag meetings where food is provided.
    For I As Integer = 0 To e.Events.Count - 1
        Dim pItem As TMS.WebPlanner.PlannerEvent = e.Events(I)
        Dim hasFood As Boolean = _
            Convert.ToBoolean(pItem.OtherColumns("FoodProvided"))
        pItem.Flagged = hasFood
        pItem.FlagColor = Color.Red
    Next I

End Sub
```

## Hints

As users schedule more and more meetings and other types of events in the `WebPlanner` or `MonthPlanner`, the planner could become crowded. In those situations, the event notes and caption may not be fully visible. The user still needs to see information about the event. The user could edit the event in order to view its information, but that can

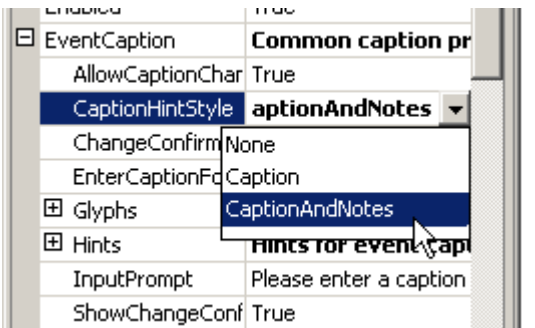
become inefficient especially if there are a number of events to be reviewed. WebPlanner provides two features that address this problem: Caption Hints and Custom Hints.

Caption Hints

By default, when a user moves their mouse over an event's caption, a tooltip or hint window appears above the caption. The hint window contains the event's topic (i.e., caption). You can turn off the caption hint entirely or have it display the event's topic and notes. To choose the information displayed in the hint window, do the following:

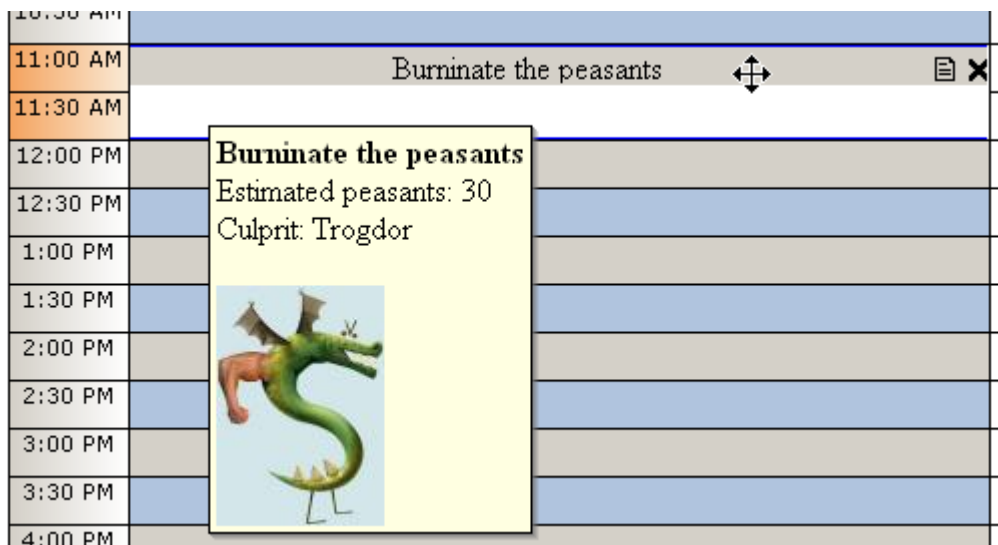
1. In Visual Studio.NET, click the planner control on your web form.
2. Go to the Properties Window and find the **EventCaption** property.
3. Expand the EventCaption property and select a value for the **CaptionHintStyle** property.

The following screenshot shows the options available for the CaptionHintStyle property.



Custom Hints

Caption hints work for many situations. However, the user may need to see application-specific information that the WebPlanner doesn't know about and cannot put into a caption hint. For example, in a volunteer scheduling application, the user may not only want to see the name of the volunteer event but a list of the volunteers or volunteer groups scheduled to attend. In those situations, the application can make use of *custom hints*. You can define a custom hint per PlannerEvent, as you see fit. The hint can contain HTML markup as well as literal text. The following screenshot shows an example of a custom hint that displays application-specific information and uses an IMG tag to display a relevant graphic.





For relevant information about Trogdor, please see <http://www.homestarrunner.com/sbemail58.html>

To set a custom hint, set a value for the PlannerEvent's **Hint** property. If Hint is null or an empty string, no custom hint is displayed. There are two points in time when the application could set custom hints. The first point in time is when the WebPlanner or MonthPlanner raises the EventHint event. The following code example shows a delegate for **EventHint**:

### C# Example

```
private void WebPlanner1_EventHint(object sender,
    TMS.WebPlanner.PlannerEventHintEventArgs e)
{
    e.Hint = "<b>Burninate the peasants</b><br>" +
        "Estimated peasants: 30<br>" +
        "Culprit: Trogdor" +
        "<p></p><img src='images/trogdor.jpg'>";
}
```

### VB.NET Example

```
Private Sub WebPlanner1_EventHint(ByVal sender As Object, _
    ByVal e As TMS.WebPlanner.PlannerEventHintEventArgs) _
    Handles WebPlanner1.EventHint

    e.Hint = "<b>Burninate the peasants</b><br>" + _
        "Estimated peasants: 30<br>" + _
        "Culprit: Trogdor" + _
        "<p></p><img src='images/trogdor.jpg'>"

End Sub
```

The EventHint event is fired once per PlannerEvent in the WebPlanner's **Items** collection. A more efficient spot to set custom hints is when the WebPlanner or MonthPlanner raises the **EventsRetrieved** event. A delegate for the EventsRetrieved event is called only once. The delegate must be coded to roll through the collection of PlannerEvents, setting a custom hint for each. The following code snippet provides an example delegate for EventsRetrieved.

### C# Example

```
using TMS.WebPlanner;
...
private void WebPlanner1_EventsRetrieved(object sender,
    PlannerCollectionEventArgs e)
{
    for (int i = 0; i < e.Events.Count; i++)
    {
        PlannerEvent anEvent = (PlannerEvent) e.Events[i];

        anEvent.Hint = String.Format("<b>{0}</b><br>" +
            "Estimated peasants: {1}<br>" +
            "Culprit: {2}" +
            "<p></p><img src='{3}'>",
            anEvent.Caption.Text,
            anEvent.OtherColumns["PeasantCount"],
            anEvent.OtherColumns["CulpritName"],
            anEvent.OtherColumns["CulpritImageUrl"]);
    }
}
```

## VB.NET Example

```

Private Sub WebPlanner1_EventsRetrieved(ByVal sender As Object, _
    ByVal e As TMS.WebPlanner.PlannerCollectionEventArgs) _
    Handles WebPlanner1.EventsRetrieved

    For I As Integer = 0 To e.Events.Count - 1

        Dim anEvent As TMS.WebPlanner.PlannerEvent = e.Events(I)

        anEvent.Hint = String.Format("<b>{0}</b><br>" + _
            "Estimated peasants: {1}<br>" + _
            "Culprit: {2}" + _
            "<p></p><img src='{3}'>", _
            anEvent.Caption.Text, _
            anEvent.OtherColumns("PeasantCount"), _
            anEvent.OtherColumns("CulpritName"), _
            anEvent.OtherColumns("CulpritImageUrl"))

        Next
    End Sub

```

By default, an event's custom hint appears when the user moves the mouse over any part of the event. If you do not want Caption Hints to also appear, be sure to set the **EventCaption.CaptionHintStyle** property to the value **None**. This property is available on both the MonthPlanner and the WebPlanner.

### Hint Positioning

By default, all custom hint windows are displayed 40 pixels to the left and 40 pixels below the top, left corner of the event's caption. To change these values, use the WebPlanner's or MonthPlanner's **EventHints** composite property. If you expand the EventHints property within the Properties Window, you'll see a number of subproperties. The **DeltaX** and **DeltaY** properties control the positioning of the custom hint window. Note that changes to these properties affect all custom hint windows displayed by the planner.

Use DeltaX to affect the hint window's horizontal position relative to the left edge of the event caption.

Use the DeltaY property to affect the hint window's vertical position relative to the top edge of the event caption.

### Hint Appearance

By default, custom hints display for 5 seconds and the hint window fades away once the time limit has expired. If the user moves the mouse to another event, the hint window from the previous hint is immediately closed and the new hint window displays. In addition, the hint window has a default background color that is set to KnownColor.Info. You can control these behaviors and appearance via the WebPlanner's or MonthPlanner's **EventHints** composite property. Note that changes to the **EventHints** composite property affect all custom hint windows.

To change the length of time the window is displayed, use the **Pause** property. Set it to the number of milliseconds the window should be displayed.

To have the hint window close immediately instead of fading away, set the **Fading** property to the value false.

To change the background color of the hint window, set the **BackColor** property to the desired color.

## DataStores

---

One of the goals of WebPlanner is to make it very easy to get data into the WebPlanner and MonthPlanner controls. Instead of using data binding, the planner controls interface with *DataStores*. DataStores encapsulate retrieval logic and event handling that couldn't be handled with data binding. Instead of making every person using WebPlanner write that kind of code, we put it all into the *IDataStore* interface and the *IDataStore* implementations. Note that DataStores are used in ASP.NET 1.1 only. In ASP.NET 2.0, the new *DataSource* components can be used.

A *DataStore* provides *PlannerEvents* to a WebPlanner. The manner in which it does so is defined by the *IDataStore* interface. However, the WebPlanner is ignorant of where and how the data store manages the *PlannerEvents*. The events could be in an XML file or in a relational database. It doesn't matter. This allows an application to switch its WebPlanner from one source of information to another via a property setting (the Meeting Rooms example installed with this product shows how to do this). It also gives other parties the freedom to develop their own data storage mechanisms. For example, *PlannerEvents* could be retrieved, inserted, updated, and deleted via a custom *WebServiceDataStore*.

**Note:** Data stores do not manage the resources that are being scheduled.

WebPlanner includes the following *DataStores*:

- *OleDbDataStore* - Allows *PlannerEvents* to be managed by any database accessible via an OLE DB driver.
- *SqlClientDataStore* - Allows *PlannerEvents* to be managed by Microsoft SQL Server.
- *XmlDataStore* - Manages *PlannerEvents* via an in-memory *DataSet* and persisted to an XML file.

The *OleDbDataStore*, *SqlClientDataStore*, and *BdpDataStore* are well suited for multi-user applications. Because the *XmlDataStore* persists its data to an XML file, it is geared more towards zero-code demonstrations and single user applications. To associate a *DataStore* with a WebPlanner, do the following:

1. Drop the appropriate data store component onto the form.
2. Click on the WebPlanner and go to the Properties page.
3. Specify the name of the data store in the WebPlanner's *DataStore* property.

Depending upon the type of data store used, additional configuration steps may be required. The following sections explain how to use the data stores listed previously.

### Managing PlannerEvents by hand

An application is not required to use a data store with its WebPlanner. It is perfectly acceptable for an application to manage its *PlannerEvents* without the help of a data store. Applications may even wish to programmatically manage certain *PlannerEvents*, such as those representing yearly events or holidays, while a *DataStore* is used to manage the bulk of the scheduled events. The WebPlanner tracks the origin of each *PlannerEvent* via its **Origin** property. Events retrieved from a data store are assigned the value *EventOrigin.DataStore* while those created manually default to the value *EventOrigin.Application*.

If an application is responsible for managing *PlannerEvents*, it needs to follow several guidelines:

1. All events are stored in the **Items** collection of the WebPlanner, regardless of origin. An application may insert events directly into the collection or via a WebPlanner utility method such as **InsertPlannerEvent**.
2. The *Items* collection is refreshed upon every page request or postback. *PlannerEvents* marked as originating from the application are left in the *Items* collection if the WebPlanner's **PreserveApplicationItems** property is set to *true*. Otherwise, they are removed from the collection with the events originating from a data store.
3. The application is responsible for persisting any changes to its *PlannerEvents*. Various events provide an opportunity to do so.

- When the WebPlanner creates a new PlannerEvent, it clones the one described by its **DefaultEvent** property. The application may do the same or create them based upon its own set of default property values.

The following table lists the events that an application should handle if it wishes to manage its own data:

User Action	WebPlanner Event
User creates new event	EventCreated
User changes an existing event	EventEdited
User deletes an event	EventDeleted ( <b>Note:</b> The event handler is responsible for removing the event from the WebPlanner.Items collection)

## Common DataStore functionality

Each data store operates under the same basic guidelines.

- All PlannerEvents are stored in the same table.
- Each PlannerEvent is identified by a unique ID. If the ID field is not an autonumber field, the data store creates a GUID for each PlannerEvent it creates.
- The table containing the PlannerEvents must have fields representing the item start time and end time.
- The table containing the PlannerEvents may also have fields for an event's subject, notes, and the ID of the resource with which it is associated.
- The table may contain other fields that are pertinent to the application. These field names and their values are stored in the PlannerEvent's **OtherColumns** property. This information is available to the application once the PlannerEvent has been retrieved via the data store.

Because the data store needs information about the table containing the PlannerEvents and the names of the fields containing specific data items, the table must be defined via a typed or untyped DataSet. To associate a data store with a DataSet, do the following:

- Create the DataSet, defining the table containing the PlannerEvents and define the fields it contains.
- Select the data store component and go to the Properties page.
- Set the value of the data store's **DataSet** property to the name of the DataSet.
- Set the value of the data store's **Table** property to the name of the table containing the PlannerEvents.
- Set the values of the data store's field-related properties as shown in the following table:

Property name	Value
EndTimeField	The name of the field containing the event's end time.
KeyField	The name of the field containing the event's unique ID.
NotesField	The name of the field containing the notes associated with the PlannerEvent.
ResourceField	The name of the field containing the unique ID of the resource with which the event is associated.
StartTimeField	The name of the field containing the event's start time.
SubjectField	The name of the field containing the event's subject or caption text.

Depending upon the type of data store used, other configuration steps may be necessary.

## SqlClientDataStore

The `SqlClientDataStore` manages `PlannerEvents` in a Microsoft SQL Server database. The events are retrieved from the database based upon the WebPlanner's current mode. When an event is inserted, updated, or deleted, the data store uses dynamic SQL statements to make the corresponding change in the database.

When retrieving events, the `SqlClientDataStore` builds a basic `SELECT` statement that grabs all events occurring within the date range appropriate for the current mode. It does not filter the events by resource or by any criteria other than the date range. An application may modify the `SELECT` statement by supplying a delegate for the **ChangeQuery** event. This event is raised just before the `SELECT` statement is executed against the database.

As mentioned in the [previous](#) section, the form must contain a typed or untyped `DataSet` describing the name and structure of the table containing the scheduled items. In addition, the form must also provide a `SqlConnection` component that is configured to connect to the correct SQL Server database. Once the `SqlConnection` has been configured, connect it to the `SqlClientDataStore` via the data store's **Connection** property.

To set up a `SqlClientDataStore` based upon a Typed `DataSet`, please do the following:

1. Create the table to hold the scheduled events in the Sql Server database. At a minimum, it needs fields for start time, end time, unique ID, resource ID, subject (i.e., what is displayed in the `PlannerEvent`'s caption), and notes.
2. The next step is to create a Typed `DataSet` that can be used by the `SqlClientDataStore` to understand the structure of the table containing the scheduled events. Drop a `SqlDataAdapter` on the form. The Data Adapter Configuration Wizard appears. Click Next to advance to the next page.
3. On the "Choose Your Connection" page, specify the connection string for the SQL Server database. Click the Next button.
4. On the "Choose a Query Type" page, select the "Use SQL statements" option. Click the Next button.
5. On the "Generate the SQL statements" page, specify a SQL statement like "select \* from <event table name>" and click the Next button.
6. On the "View Wizard Results" page, the wizard may indicate that it could not create `UPDATE` or `DELETE` statements. Ignore this and click the Finish button.
7. In the IDE, right click the `SqlDataAdapter` and choose the "Generate DataSet..." menu item. A dialog titled "Generate DataSet" appears.
8. Check the "New" option and specify a name for the `DataSet` to be generated.
9. Make sure the event table is checked in the listbox and that the "Add this dataset to the designer" option is also checked. Click the OK button. The `DataSet` is created and added to the form. A `SqlConnection` is also created at the same time.
10. Go to the `SqlClientDataStore` properties. Set its `Connection` property to the generated `SqlConnection` and set its `DataSet` property to the dataset generated via the `SqlDataAdapter`.
11. At this point, specify values for the `Table` and `xxxField` properties of the `SqlClientDataStore`.
12. The `SqlDataAdapter` is not needed by the `SqlClientDataStore`. If the `SqlDataAdapter` is not needed by the application then it may be removed from the form.

Note: the `SqlClientDataStore` is not available for ASP.NET 2.0. For ASP.NET 2.0, a `SqlClientDataSource` can be used to connect directly to the WebPlanner in an identical way as it can be connected to a `GridView`.

## OleDbDataStore

The `OleDbDataStore` manages `PlannerEvents` in any database accessible via an OLE DB driver. For example, the back end database could be implemented using Microsoft Access. The events are retrieved from the database based upon the WebPlanner's current mode. When an event is inserted, updated, or deleted, the data store uses dynamic SQL statements to make the corresponding change in the database.

When retrieving events, the OleDbDataStore builds a basic SELECT statement that grabs all events occurring within the date range appropriate for the current mode. It does not filter the events by resource or by any criteria other than the date range. An application may modify the SELECT statement by supplying a delegate for the **ChangeQuery** event. This event is raised just before the SELECT statement is executed against the database.

As mentioned in the [Common DataStore functionality](#) section, the form must contain a typed or untyped DataSet describing the name and structure of the table containing the scheduled items. In addition, the form must also provide an OleDbConnection component that is configured to connect to the correct database. Once the OleDbConnection has been configured, connect it to the OleDbDataStore via the data store's **Connection** property.

To set up an OleDbDataStore based upon a Typed DataSet, please do the following:

1. Create the table to hold the scheduled events in the database. At a minimum, it needs fields for start time, end time, unique ID, resource ID, subject (i.e., what is displayed in the PlannerEvent's caption), and notes.
2. The next step is to create a Typed DataSet that can be used by the OleDbDataStore to understand the structure of the table containing the scheduled events. Drop an OleDbDataAdapter on the form. The Data Adapter Configuration Wizard appears. Click Next to advance to the next page.
3. On the "Choose Your Connection" page, specify the connection string for the database. Click the Next button.
4. On the "Choose a Query Type" page, select the "Use SQL statements" option. Click the Next button.
5. On the "Generate the SQL statements" page, specify a SQL statement like "select \* from <event table name>" and click the Next button.
6. On the "View Wizard Results" page, the wizard may indicate that it could not create UPDATE or DELETE statements. Ignore this and click the Finish button.
7. In the IDE, right click the OleDbDataAdapter and choose the "Generate DataSet..." menu item. A dialog titled "Generate DataSet" appears.
8. Check the "New" option and specify a name for the DataSet to be generated.
9. Make sure the event table is checked in the listbox and that the "Add this dataset to the designer" option is also checked. Click the OK button. The DataSet is created and added to the form. An OleDbConnection is also created at the same time.
10. Go to the OleDbDataStore properties. Set its Connection property to the generated OleDbConnection and set its DataSet property to the dataset generated via the OleDbDataAdapter.
11. At this point, specify values for the Table and xxxField properties of the OleDbDataStore.
12. The OleDbDataAdapter is not needed by the OleDbDataStore. If the OleDbDataAdapter is not needed by the application then it may be removed from the form.

Note: the OleDbClientDataStore is not available for ASP.NET 2.0. For ASP.NET 2.0, an OleDbDataSource can be used to connect directly to the WebPlanner in an identical way as it can be connected to a GridView.

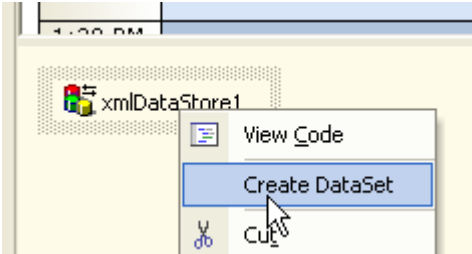
## XmlDataStore

The XmlDataStore uses a DataSet to hold PlannerEvents in memory and persists the events to an XML document. It is ideally suited for creating prototypes and zero-code WebPlanner demonstrations. It is also suited for single user applications. If an application will support more than one user or needs strict transactional control over its data, use the [SqlClientDataStore](#) or [OleDbDataStore](#) instead. The DataSet used by the data store may be created automatically or manually at design time, as described in the following sections.

Note: the XmlDataStore is not available for ASP.NET 2.0. As the Microsoft XmlDataSource equivalent is a non-updatable implementation, Xml data storage can not be used in ASP.NET 2.0

### Auto-creating a DataSet

The XmlDataStore provides a "Create DataSet" feature that, at design time, programmatically creates a DataSet compatible with the datastore and connects the DataSet to the datastore. To create the DataSet, right click the XmlDataStore and select "Create DataSet" from the popup menu.



The XmlDataStore designer creates a new DataSet component on the form, creates a table within the DataSet, and adds the appropriate fields to the table. By default the XmlDataStore will automatically save the table's records to an XML file named Events.xml. The file will be located in the directory containing the form. Once the data store is connected to the WebPlanner, the WebPlanner is able to read from and write to the XML file via the XmlDataStore.

**Manually creating an Untyped DataSet**

A custom, hand crafted DataSet may also be connected to the DataStore. The WebPlanner has few requirements regarding how the scheduled events are stored in the DataSet. The requirements are as follows:

1. The scheduled events must be stored in a single table.
2. The events must be uniquely identified via some type of key field.
3. Each scheduled event must have a start time and an end time.

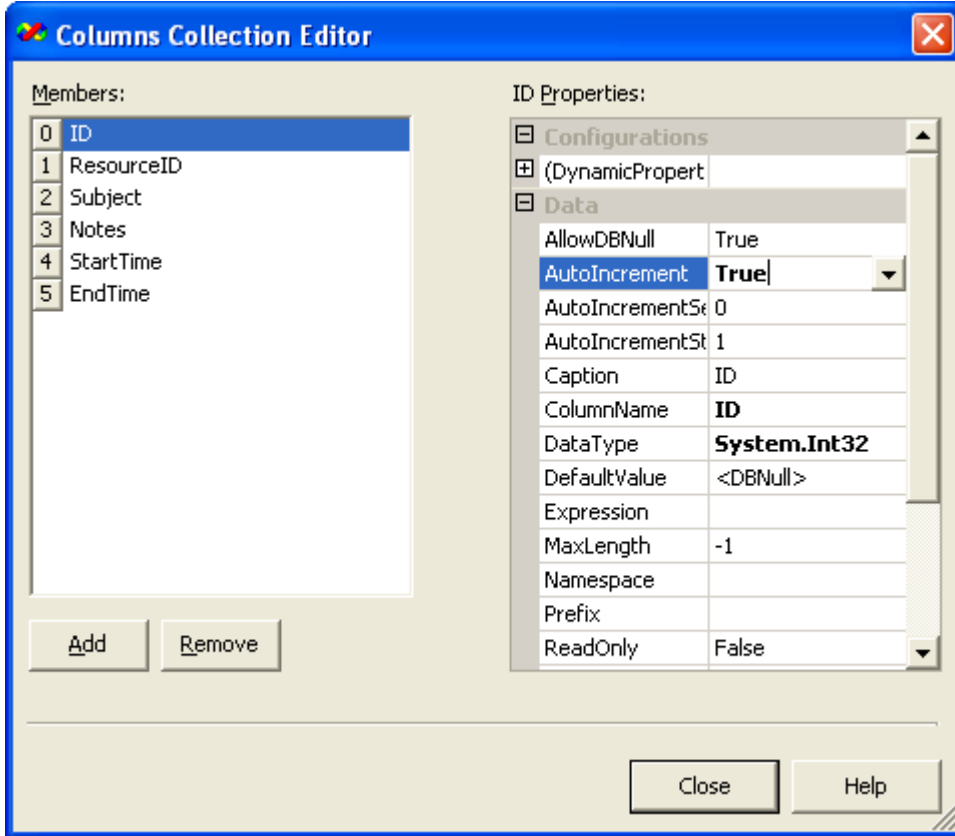
Other than those requirements, the WebPlanner does not care what the table or its fields are named. To set up the DataSet, do the following:

1. From the Data tab of the Visual Studio.NET Toolbox, drop a DataSet component onto the form. When prompted to create either a Typed or Untyped DataSet, choose Untyped.
2. Select the DataSet, go to the Properties page, and change the name of DataSet from *DataSet1* to something more appropriate.
3. In the Properties page, open the collection editor for the DataSet's Table property.
4. In the Table collection editor, add a new table and change its name appropriately.
5. In the properties for the table, open the collection editor for the Columns properties.
6. Add the following columns to rides.

Column name	Data Type	Notes
ID	System.Int32	This is the unique identifier for a scheduled event. Set the AutoNumber property to the value true. If autonumbering is not desired, it is also permissible to use types System.String or System.GUID. For either of those field types, the DataStore will generate a GUID to serve as the key for each item.
ResourceID	System.Int32	The unique ID of the resource with which the scheduled event is associated.
Subject	System.String	The information displayed in the caption of the event when displayed on the WebPlanner.
Notes	System.String	Miscellaneous information associated with the event. For example, this could contain the reason why the event has been scheduled.
StartTime	System.DateTime	The start time of the event.

EndTime	System.DateTime	The end time of the event.
---------	-----------------	----------------------------

When you have finished, the Columns collection editor should appear as follows:

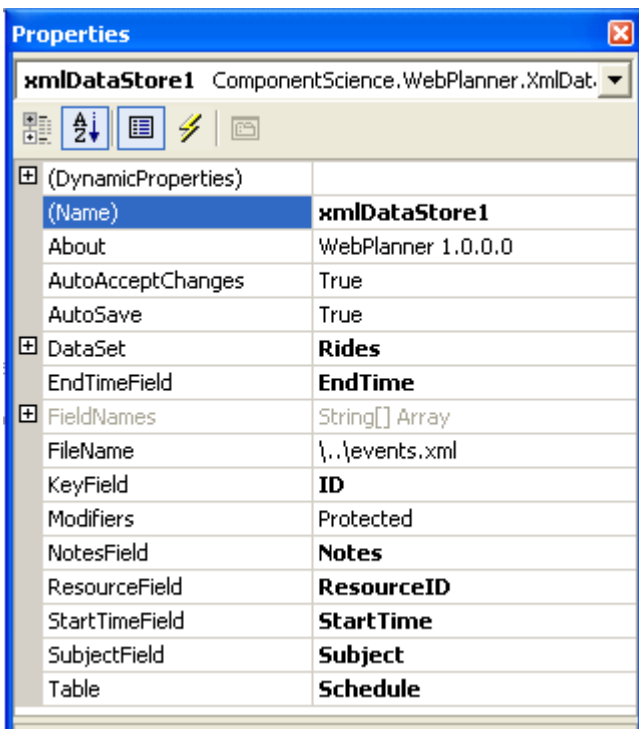


Commit your changes by clicking the Close button on the Columns Collection Editor followed by clicking the Close button on the Tables Collection Editor. Once you've saved your project, select the XmlDataStore and configure its properties as follows and in the order shown:

Property	Value	Notes
DataSet	<your DataSet>	This is the DataSet used by the XmlDataStore to read and write data.
Table	<table name>	This is the table within the DataSet where the XmlDataStore reads and writes scheduled events.
EndTimeField	EndTime	The field containing the end time of the event.
KeyField	ID	The field within the table containing the unique ID for each event.
NotesField	Notes	The notes field.
ResourceField	ResourceID	The field containing the unique ID of the resource whose time has been scheduled.
StartTimeField	StartTime	The field containing the start time of the event.
SubjectField	Subject	The field containing the information displayed in the caption of the event on the WebPlanner.

When finished, the XmlDataStore properties should appear as follows:





At this point in the project, the WebPlanner is able to read and write information via the XmlDataStore.

## Data persistence

By default, the XmlDataStore automatically accepts changes made to the DataSet and saves the changes to an XML file. Every time a PlannerEvent is added, changed, or deleted, the XML file is written. Depending upon the amount of data, the disk I/O may become noticeably slow. To have the application control the persistence of data itself, set the XmlDataStore's **AutoSave** property to the value *false*. The remainder of this section describes how the application can control the reading and writing of PlannerEvents via event handlers and the DataSet's **ReadXml** and **WriteXml** methods.

To read and write the XML document, code must be added to four different event handlers. To read the data from the XML file, insert the following code into the application's Page\_Load event handler. Note that the code is simplified for this example and could be encapsulated into stand alone methods for a real life application.

### C# Page\_Load

```
using System.IO; // Add this to the set of using clauses at page top

private void Page_Load(object sender, System.EventArgs e)
{
    string xmlDataFile = "c:\\temp\\Rides.xml";
    if (System.IO.File.Exists(xmlDataFile))
        RidesDataSet.ReadXml(xmlDataFile);
}
```

### VB.NET Page\_Load

```
Private Sub Page_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
    Dim xmlDataFile As String
```

```

xmlDataFile = "c:\temp\Rides.xml"
If (System.IO.File.Exists(xmlDataFile)) Then
    RidesDataSet.ReadXml(xmlDataFile)
End If
End Sub

```

This code in the Page\_Load event looks for a file named Rides.xml in directory C:\temp. If it exists then it loads the XML file into the dataset. If such a file existed and we ran the application, an error would occur. That is because the application runs under the auspices of the ASPNET user account and that account does not have access to the file system on your computer. Before we can run the application, ASPNET must be granted read and write permissions to C:\temp.

To grant permissions to ASPNET user, do the following:

1. Open Windows Explorer and go to the C:\temp directory. Create it if it does not exist.
2. Select the Tools→Options menu item and go to the View page.
3. In the Advanced Settings listbox of the View page, scroll down to the bottom.
4. Make sure the **Use simple file sharing option** is not checked.
5. Click the OK button.
6. Right click the C:\temp folder in Windows Explorer and choose the *Sharing and Security* item from the popup menu.
7. Go to the Security page.
8. If the ASPNET user is not listed, click the Add button.
9. In the textbox labeled "Enter the object names to select, enter the text **<computer name>\ASPNET** and click the OK button.
10. With ASPNET user selected, assign it Read and Write permissions in the Permissions listbox.
11. Click the OK button.

With that out of the way, add the following code for the following WebPlanner event handlers.

## C# Delegates

### WebPlanner.EventDeleted

```

private void WebPlanner1_EventDeleted(object sender, TMS.WebPlanner.PlannerEventEventArgs e)
{
    RidesDataSet.WriteXml("c:\\temp\\Rides.xml");
}

```

### WebPlanner.EventInserted

```

private void WebPlanner1_EventInserted(object sender, TMS.WebPlanner.PlannerEventEventArgs e)
{
    RidesDataSet.WriteXml("c:\\temp\\Rides.xml");
}

```

### WebPlanner.EventUpdated

```

private void WebPlanner1_EventUpdated(object sender, TMS.WebPlanner.PlannerEventEventArgs e)
{
    RidesDataSet.WriteXml("c:\\temp\\Rides.xml");
}

```

## VB.NET Delegates

**WebPlanner.EventDeleted**

```
Private Sub WebPlanner1_EventDeleted(ByVal sender As Object, ByVal e As  
TMS.WebPlanner.PlannerEventEventArgs) Handles WebPlanner1.EventDeleted  
    RidesDataSet.WriteXml("c:\temp\Rides.xml");  
End Sub
```

**WebPlanner.EventInserted**

```
Private Sub WebPlanner1_EventInserted(ByVal sender As Object, ByVal e As  
TMS.WebPlanner.PlannerEventEventArgs) Handles WebPlanner1.EventInserted  
    RidesDataSet.WriteXml("c:\temp\Rides.xml");  
End Sub
```

**WebPlanner.EventUpdated**

```
Private Sub WebPlanner1_EventUpdated(ByVal sender As Object, ByVal e As  
TMS.WebPlanner.PlannerEventEventArgs) Handles WebPlanner1.EventUpdated  
    RidesDataSet.WriteXml("c:\temp\Rides.xml");  
End Sub
```

These event handlers are called when a planner event is deleted, inserted, or updated. Just like the XmlDataStore's auto saving feature, this implementation makes sure that all of the changes are saved by writing the XML file each time a change occurs. If the number of records grows large enough, this amount of disk I/O could affect performance.

# Recurrency

---

This chapter discusses how the WebPlanner internally handles recurring events and explains the helper class PlanRecurr to manipulate and edit recurring events.

The WebPlanner recurrency formulas are based on the vCalendar spec. Information about the recurrency specifiers can be found at <http://www.imc.org/pdi/vcal-10.txt>

## Database requirements

WebPlanner has built-in recurrency handling. That means that a single record in the database can have a recurrency specifier and this will result in a series of recurrent events displayed in the WebPlanner. Multiple Planner events can as such refer to the same database record. When one of the events is changed on the WebPlanner, the record will be updated and as a consequence of this, all events in the series will be updated as well.

To handle recurrency, 3 extra fields in the table of events are required:

**RecurrencyField** : string field holding the recurrency formula. The WebPlanner supports the vCalendar standard for recurrency specifications where applicable. That means it supports the standard for recurrency specifications that can be handled in the WebPlanner. For example, a by minute recurrency is not supported as this doesn't make much sense for the WebPlanner time views.

**FirstStartTimeField** : this field is a DateTime field holding the start time of the first event in the series. Note that **StartTimeField** now holds the start time of the first event in the recurrent series.

**FirstEndTimeField** : this field is a DateTime field holding the end time of the first event in the series. Note that **EndTimeField** now holds the end time of the last event in the recurrent series.

When these 3 extra fields are available in the table, specify the fields in the DataStores field properties for ASP.NET 1.1 or in the WebPlanner field properties for ASP.NET 2.0.

## Exceptions

The recurrency formulas also support exceptions. That means that an exception date/time can be set and that on this exception date/time no recurrent event will be scheduled. The exceptions are part of the recurrency formula and stored in the RecurrencyField of the database table.

By default moving, sizing, editing one event in a recurrent series in the WebPlanner will change the time or properties of all events in the recurrent series. If one event in the series is moved to one day later, all events in the recurrent series will be moved to one day later. If the duration of one event in the series is made one hour longer, all events in the series will be one hour longer. Or when the subject for one event is changed, all subjects for recurrent events are changed.

When this behaviour is not desired and instead of changing all events of the series an exception should be created, this can be done automatically by the WebPlanner. To enable this, set

**WebPlanner.Alerts.ConfirmRecurrencyException** to true. When a change is made to one event in a series, WebPlanner will prompt to create an exception or not. The message that is displayed in the prompt is set by **WebPlanner.Alerts.ConfirmRecurrencyExceptionMessage**. When it is accepted that an exception will be created, the WebPlanner takes following steps:

1. It modifies the formula of the recurrent series to insert an exception at the time of the edited event and updates the record in the database holding the recurrent series.
2. It creates a new event for the event that was edited

Note that from the time the exception event has been created, editing this new exception event no longer affects the recurrent series.

## Editing recurrency

To facilitate the editing of recurrencies, a helper class has been created that is able to parse and decompose a recurrency formula and generate a formula as well. This helper class is `TMS.WebPlanner.PlanRecurr`.

The `PlanRecurr` class has following properties:

### **RecurrencyFrequency Frequency**

This specifies the recurrency frequency. Possible values are:  
Hourly, Daily, Weekly, Monthly, Yearly, None

### **int Interval**

Sets the interval between recurrent events. By default, this is 1. For an hourly recurrency with `Interval` set to 1, this means that there is one hour between two consecutive events in the recurrent series. For a daily recurrence with `Interval` set to 2, this means there are 2 days between two consecutive events in the recurrent series.

### **int RepeatCount**

Sets the number of events in the recurrent series. The end of a recurrent series is defined by either a `RepeatCount` or a value for `RepeatUntil`. When neither `RepeatCount` or `RepeatUntil` are defined, the recurrent series is infinite. When `RepeatCount` is 0, it is considered not defined.

### **DateTime RepeatUntil**

Sets the end of the recurrent series as a date. The end of a recurrent series is defined by either a `RepeatCount` or a value for `RepeatUntil`. When neither `RepeatCount` or `RepeatUntil` are defined, the recurrent series is infinite. When `RepeatUntil` is a null date, it is considered not defined.

### **DateTime MinTime**

Read-only property returning the start time of the first event in a recurrent series.

### **DateTime MaxTime**

Read-only property returning the end time of the last event in a recurrent series.

### **ArrayList ExDates**

This is an arraylist of exception dates. The arraylist contains objects of the type `SEDate` that holds a start and end time for an exception. `SEDate` has only two properties:

```
DateTime SEdate.StartDate  
DateTime SEdate.EndDate
```

### **ArrayList Dates**

This is an arraylist of all start/end times of all events in the recurrent series. The arraylist contains objects of the type `SEDate` that holds a start and end time for an event.

Example: reading all start and end times of events in a series

```
foreach(SEDate se in PlanRecurr.Dates)
{
    Trace.Write("Event from : " + se.StartDate + " to "+ se.EndDate);
}
```

### string Days

This is a string holding day specifiers for days where the event is recurring. The day specifiers are the first two letters of english day names in capital letters, ie. MO, TU, WE, TH, FR, SA, SU.

If an event should repeat every Tuesday and Thursday, the Days property can be set to "TUTH"

### int [] DayNum

This is an array of 7 integer values that optionally specify on which day of the month an event is recurring.

DayNum[0] : is a value between 0 & 3 defining which sunday in the month the event is recurring

DayNum[1] : is a value between 0 & 3 defining which monday in the month the event is recurring

...

DayNum[6] : is a value between 0 & 3 defining which saturday in the month the event is recurring

### Example: Getting information of a parsed recurrency formula with PlanRecurr

```
PlanRecurr.Parse("RRULE:FREQ=BYDAY;INTERVAL=1;COUNT=5");
```

This parses the recurrency into :

```
PlanRecurr.Frequency = Daily
```

```
PlanRecurr.Interval = 1
```

```
PlanRecurr.RepeatCount = 5
```

### Example: Getting all dates of a recurrent series

```
PlanRecurr.StartTime = new DateTime(2005,10,25,10,0,0);
```

```
PlanRecurr.EndTime = new DateTime(2005,10,25,12,0,0);
```

```
PlanRecurr.Generate("RRULE:FREQ=BYDAY;INTERVAL=1;COUNT=5");
```

This will fill the PlanRecurr.Dates arraylist with SEDate objects holding start / end times of the recurrent events

Example: Composing a recurrency formula

```
PlanRecurr.Frequency = Daily
```

```
PlanRecurr.Interval = 1
```

```
PlanRecurr.RepeatCount = 5
```

```
Trace.Write(PlanRecurr.Compose());
```

This will show the generated formula: "RRULE:FREQ=BYDAY;INTERVAL=1;COUNT=5"

In the PDCPlanner example, a simple recurrency editor that uses the PlanRecurr helper class is provided that shows how recurrency can be edited.

When a recurrency is defined for an event, this can be indicated in the event caption. By default this displays a small circular arrow symbol but this symbol can be customized with the property

**WebPlanner.EventCaption.Glyphs.RecurrentEvent**. When this symbol is clicked, it can trigger the WebPlanner server delegate EventRecurrencyEdit from where a recurrency editing form can be shown. When

WebPlanner.EditType is set to DetailEmbedded, this will be displayed in a dialog in the same browser window as the WebPlanner otherwise it will open a new browser window. The form (ASPX file) that is opened for recurrency editing is set by **WebPlanner.DetailEdit.RecurrencyTarget**. For more information about detail editing, see [Detail Edit mode](#)

## Using ClientEvents

---

Sometimes it is desirable to interact with the WebPlanner client-side using Javascript. For this purpose, the WebPlanner offers different ClientEvents. ClientEvents are properties that allow to insert Javascript code that will be executed when a given event occurs client-side in the WebPlanner.

Following such client events are available:

**CellDoubleClick** : executed when a WebPlanner cell is double clicked

Parameters:

Row : row index of the cell

Col : column index of the cell

**CellRightClick** : executed when a WebPlanner cell is right clicked

Parameters:

Row : row index of the cell

Col : column index of the cell

**CellsSelected** : executed after a selection of WebPlanner cells ends

Parameters:

SelectStart : index of first selected cell along the time axis

SelectEnd : index of last selected cell along the time axis

SelectPosition : index along the resource/day axis of the selected cells

**EditDone** : executed after inplace or popup editing ends

No parameters

**EventCustomClick** : executed when a WebPlanner event caption custom button is clicked

Parameters:

EventID : Id of the event

EventKey : key value of event

**EventDoubleClick** : executed when a WebPlanner event is double clicked

Parameters:

EventID : Id of the event

EventKey : key value of event

**EventDrag** : executed during dragging of an event in the WebPlanner

Parameters:

EventID : Id of the event

EventKey : key value of event

EventStartTime : start time of the event

EventEndTime : end time of the event

EventPosition : position index of the event

EventCaptionText : caption text of the event

EventNotesText : notes text of the event  
EvenRecurrency : recurrency rule of the event  
EventHTMLID : html id of the event  
X,Y : page coordinates of mouse

**EventDrop** : executed when an event is dropped in the WebPlanner

Parameters:

EventID : Id of the event  
EventKey : key value of event  
EventStartTime : start time of the event  
EventEndTime : end time of the event  
EventPosition : position index of the event  
EventCaptionText : caption text of the event  
EventNotesText : notes text of the event  
EvenRecurrency : recurrency rule of the event  
EventHTMLID : html id of the event  
X,Y : page coordinates of mouse

**EventDroppedInPlanner** : executed when event was dropped on WebPlanner to allow specifying whether the event is dropped inside or outside the WebPlanner (to be used when WebPlanner is partially visible)

Parameters:

EventID : Id of the event  
EventKey : key value of event  
X,Y : page coordinates of mouse

**EventMouseDown** : executed when on mousedown on event in WebPlanner

Parameters:

EventID : Id of the event  
EventKey : key value of event  
EventStartTime : start time of the event  
EventEndTime : end time of the event  
EventPosition : position index of the event  
EventCaptionText : caption text of the event  
EventNotesText : notes text of the event  
EvenRecurrency : recurrency rule of the event  
EventHTMLID : html id of the event

**EventMouseOut** : executed when on mouseout on event in WebPlanner

Parameters:

EventID : Id of the event  
EventKey : key value of event  
EventStartTime : start time of the event  
EventEndTime : end time of the event  
EventPosition : position index of the event  
EventCaptionText : caption text of the event  
EventNotesText : notes text of the event  
EvenRecurrency : recurrency rule of the event  
EventHTMLID : html id of the event



**EventMouseOver** : executed when mouse hovers over event in WebPlanner

Parameters:

EventID : Id of the event  
EventKey : key value of event  
EventStartTime : start time of the event  
EventEndTime : end time of the event  
EventPosition : position index of the event  
EventCaptionText : caption text of the event  
EventNotesText : notes text of the event  
EventRecurrency : recurrency rule of the event  
EventHTMLID : html id of the event

**EventResized** : executed when a WebPlanner event has been resized

Parameters:

EventID : Id of the event  
EventKey : key value of event  
EventStartTime : start time of the event  
EventEndTime : end time of the event  
EventPosition : position index of the event  
EventCaptionText : caption text of the event  
EventNotesText : notes text of the event  
EventRecurrency : recurrency rule of the event  
EventHTMLID : html id of the event  
resizeTo : cell index of the new end cell of the event

**EventRightClick** : executed on right-click on an event in WebPlanner

Parameters:

Parameters:  
EventID : Id of the event  
EventKey : key value of event  
EventStartTime : start time of the event  
EventEndTime : end time of the event  
EventPosition : position index of the event  
EventCaptionText : caption text of the event  
EventNotesText : notes text of the event  
EventRecurrency : recurrency rule of the event  
EventHTMLID : html id of the event

**HeaderClick** : executed when a WebPlanner header cell has been clicked

Parameters:

Position: position index of the header cell

**HeaderRightClick** : executed when a WebPlanner header cell has been right clicked

Parameters:

Position: position index of the header cell

**SideBarClick** : executed when a WebPlanner sidebar cell has been clicked

Parameters:

Position: row index of the sidebar cell

**SideBarRightClick** : executed when a WebPlanner sidebar cell has been right clicked

Parameters:

Position: row index of the sidebar cell

**WaitListEventDropped** : executed when event from WaitList is dropped on WebPlanner or MonthPlanner

Parameters:

EventID : Id of the event

EventKey : key value of event

X,Y : page coordinates of mouse

Sample:

To show the ID of a WebPlanner event that is clicked in the browser status bar without server postback, following Javascript code could be added to EventMouseDown:

```
Alert( EventID);
```

## Built-in User rights management

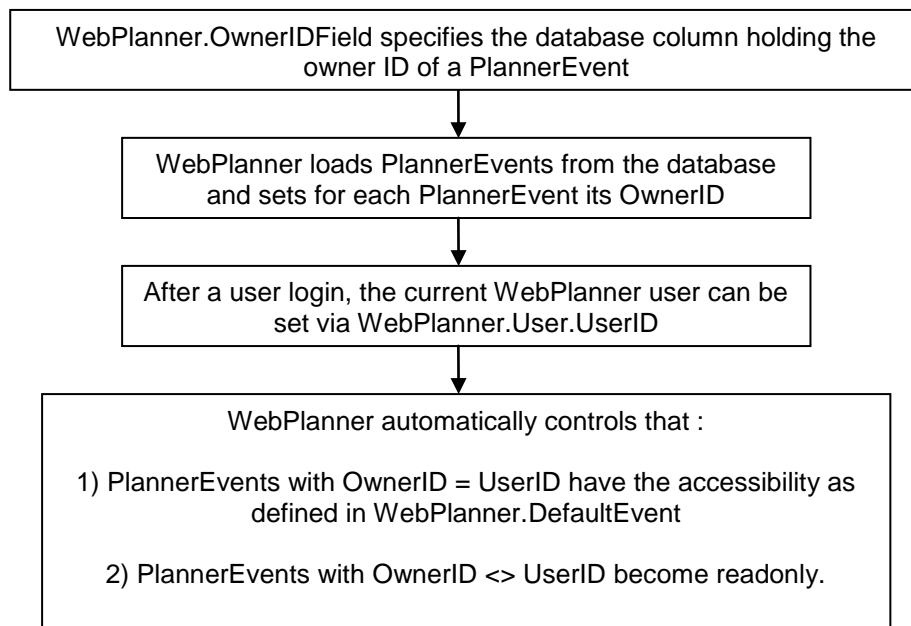
---

### Rights on PlannerEvents

In many scenarios, the WebPlanner is used by multiple users with a login for each user. A common requirement for a multiple user scenario is that users can only edit, delete the events they own. Sometimes, an additional requirement is that some resources are only available to users that 'own' the resource. Other than this, typically, one or more users are assigned administrator rights allowing administrators to edit, delete all events of the WebPlanner.

User rights management in the WebPlanner is handled by the `PlannerEvent.OwnerID` property and `WebPlanner.OwnerIDField` property for automatic databinding of the `PlannerEvent.OwnerID` to a database column. This way, each `PlannerEvent` can be owned by a user identified by `OwnerID` and will become readonly for all other users. If no `OwnerID` is assigned to an event, all users have full access to the `PlannerEvent`. The actual user of the WebPlanner is set through `WebPlanner.User.UserID`.

Schematic:



For users that need access to all WebPlanner events, it is possible to define a user as administrator. This is done by setting `WebPlanner.User.Admin` to true. When `WebPlanner.User.Admin` is true, the `PlannerEvent` access capabilities of `PlannerEvents` with an `OwnerID` different from the `WebPlanner.User.UserID` are controlled by :

For `PlannerEvents` not owned:

`WebPlanner.User.AdminRights.Edit` : allow editing events  
`WebPlanner.User.AdminRights.MoveTime` : allow to move the event along the time axis  
`WebPlanner.User.AdminRights.MovePosition` : allow to move the event along the resource axis  
`WebPlanner.User.AdminRights.Size` : allow to change the duration of the event  
`WebPlanner.User.AdminRights.Delete` : allow to delete the event  
`WebPlanner.User.AdminRights.Select` : allow to select the event  
`WebPlanner.User.AdminRights.Insert` : allow to insert a new event

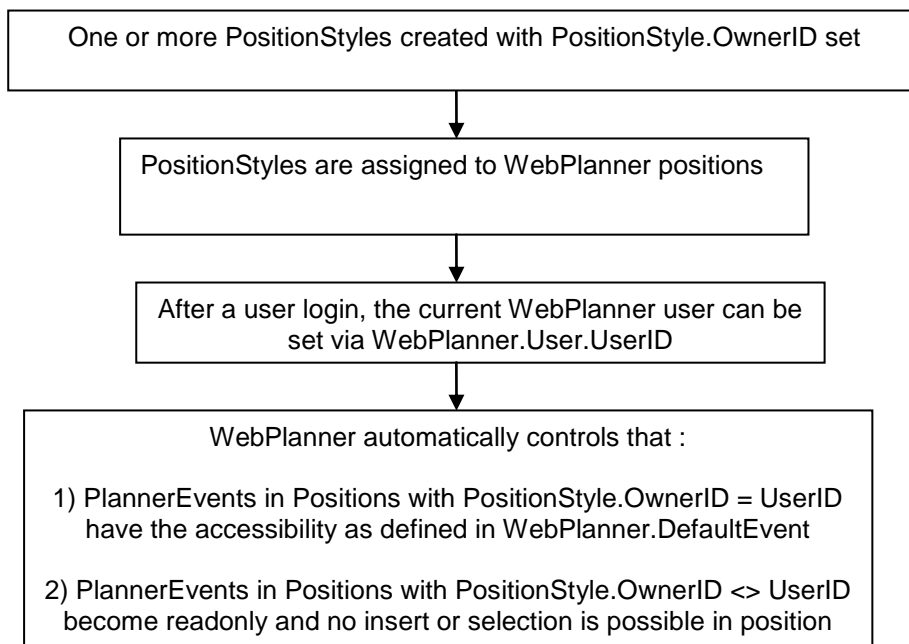
For PlannerEvents owned by the user that is administrator, ie. `PlannerEvent.OwnerID = WebPlanner.User.UserID`, the default settings apply as defined in `WebPlanner.DefaultEvent` or as customized from for example the event `WebPlanner.EventRetrieved`

Note that the use of user rights management is fully optional. When `PlannerEvent.OwnerID` or `WebPlanner.OwnerIDField` is not used, all events are always fully accessible with restrictions set by its properties (for example when dynamically set from the `EventRetrieved` event)

## Rights on Planner positions

It is equally possible to assign an ownership to a position (resource) in the WebPlanner. This can be achieved using the `PositionStyles` (see `WebPlanner.PositionStyles` collection). A `PositionStyle` can be created, added to the `WebPlanner.PositionStyles` collection and assigned to one or more positions via `WebPlanner.Positions`. The owner of a position can be set via `PositionStyle.OwnerID`. When the `WebPlanner.User.UserID` matches the `OwnerID` set to a positionstyle of a `Position`, the user can select, insert, delete, edit all events in these positions. A positionstyle with a specific `OwnerID` can be assigned to multiple positions. For positions where the positionstyle is not defined or where the `OwnerID` of the positionstyle is not defined, either the `PlannerEvent.OwnerID` determines the rights or access is given to all events in this WebPlanner position.

Schematic:



Note that the use of user rights management via `PositionStyles` is fully optional. When no `PositionStyles` are used or `PositionStyle.OwnerID` is not used, all events are always fully accessible with restrictions set by its properties (for example when dynamically set from the `EventRetrieved` event)

## WaitList

---

The WaitList component is a separate component that represents an unbound list of events. The WaitList typically holds unscheduled events. Drag & drop support is available to drag unscheduled events from the WaitList to the WebPlanner/MonthPlanner and vice versa.

### WaitList organization

The WaitList shows a list of unscheduled Planner events. It displays the events itself in an identical way as the WebPlanner or MonthPlanner in a vertical list. Events can be dragged from the WaitList to the WebPlanner and vice versa. When a drop happens, the delegate WaitListEventDropped is triggered on WebPlanner and the delegate PlannerEventDropped is triggered on the WaitList. This delegate returns the index of the Event in the WaitList items list or the WebPlanner / MonthPlanner items list.

WaitList has a DefaultEvent property that holds all the default settings for events added on the WaitList. The DefaultEvent property works in an identical way as the DefaultEvent for WebPlanner. The events itself in the WaitList are stored in the Items collection.

#### **Code example:**

This is the typical code in the WebPlanner.WaitListEventDropped delegates and WaitList.PlannerEventDropped delegates for drag & drop operation between WebPlanner & WaitList.

*Put the event dragged from the WaitList on the WebPlanner:*

```
protected void WebPlanner1_WaitlistEventDropped(object sender, TMS.WebPlanner.WaitlistEventDroppedEventArgs e)
{
    TMS.WebPlanner.PlannerEvent pe = new TMS.WebPlanner.PlannerEvent();
    TMS.WebPlanner.PlannerEvent wpe;

    // retrieve the dragged event from the WaitList
    wpe = (TMS.WebPlanner.PlannerEvent)WaitList1.Items[e.EventIndex];
    wpe.CopyTo(pe); // copy visual properties

    pe.Notes = wpe.Notes;
    pe.Subject = wpe.Subject;

    // set the location of the event on the WebPlanner
    pe.Position = e.ToPos;
    pe.BeginCell = e.ToBegin;
    pe.EndCell = e.ToBegin + 1;
    pe.Owner = WebPlanner1;
    pe.ShowViewTarget = wpe.ShowViewTarget;

    pe.StartTime = WebPlanner1.GetEventTime(pe, TimeBoundaries.Start);
    pe.EndTime = WebPlanner1.GetEventTime(pe, TimeBoundaries.End);
    pe.ResourceID = e.ToPos;
    pe.UpdateTimes();

    // insert in the datasource
    WebPlanner1.InsertPlannerEvent(pe);
}
```

*Put the event dragged from the WebPlanner in the WaitList:*

```
protected void WaitList1_PlannerEventDropped(object sender, TMS.WebPlanner.WaitlistEventDroppedEventArgs e)
```

```
{
  TMS.WebPlanner.PlannerEvent pe = new TMS.WebPlanner.PlannerEvent();
  TMS.WebPlanner.PlannerEvent wpe;

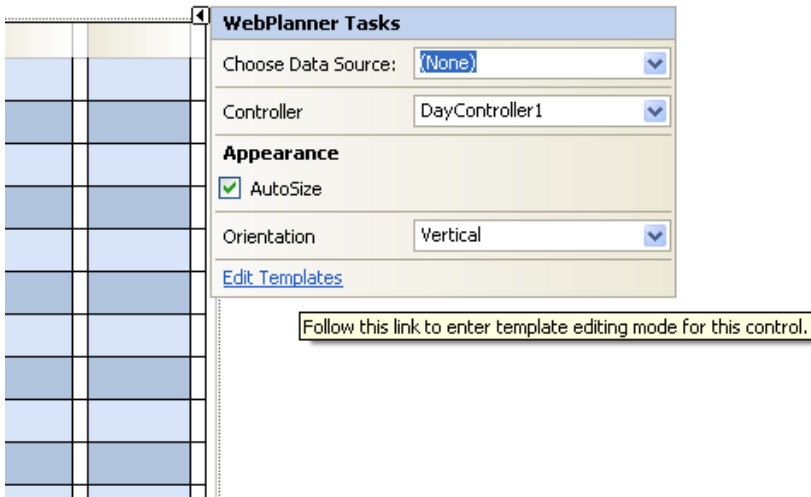
  // retrieve the dragged event from the WebPlanner
  wpe = (TMS.WebPlanner.PlannerEvent)WebPlanner1.Items[e.EventIndex];
  wpe.CopyTo(pe); // copy visual properties

  pe.Notes = wpe.Notes;
  pe.Subject = wpe.Subject;
  pe.Owner = WaitList1;
  WaitList1.Items.Add(pe);
}
```

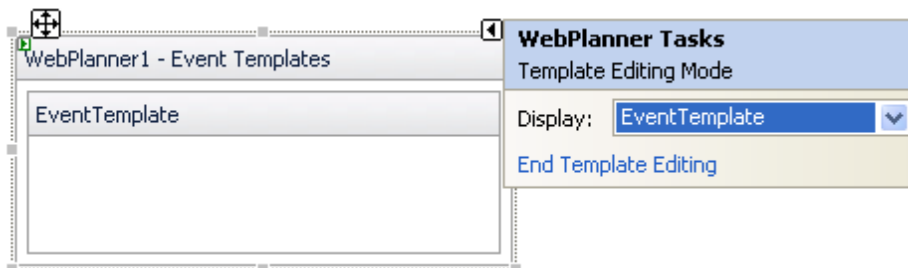
# Using Templates

The WebPlanner has full support for using templates for events with .NET 2.0. For the WebPlanner, a template can be defined for events in normal state and for events in editing state. Through templates it is possible to define at design-time the layout (with any control available in the toolbox) of an event on the WebPlanner when it is in normal state and a different template for the editing state.

To start editing templates, click on the smart tag button of the WebPlanner at design time and choose "Edit Templates":



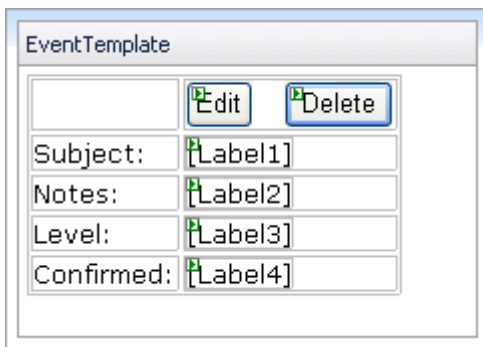
This starts the template editor for the WebPlanner and two templates can be chosen:



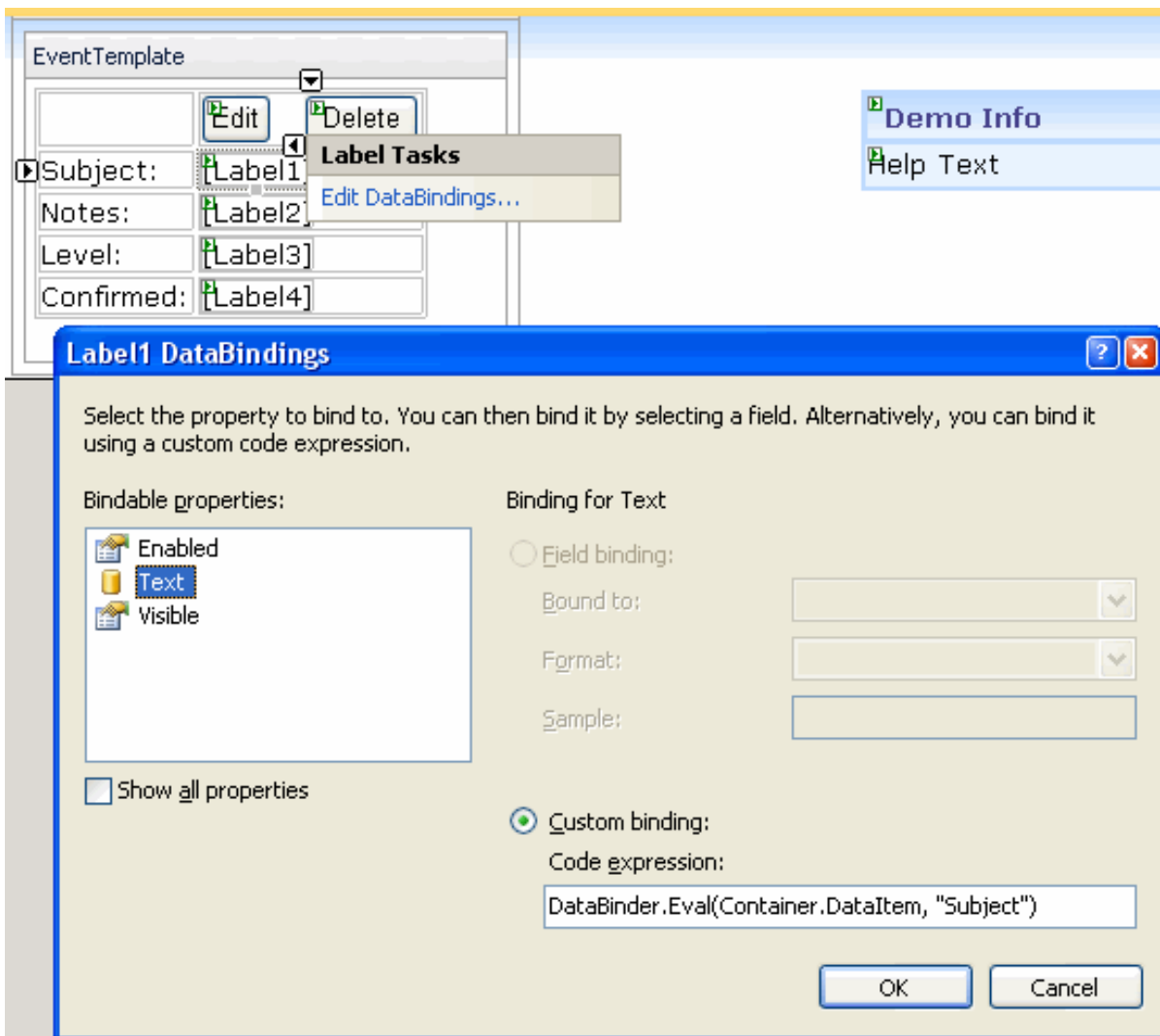
the EventTemplate (for a WebPlanner event in normal view mode) and the EditEventTemplate. (for a WebPlanner event in edit mode)

## Setting up the template

From the design time template editor, controls can be dropped on the template and optionally, a databinding can be done between these controls and the dataview that is used.



To do this, select from the smart tag for the controls dropped on the template "Edit Databindings" and through the dialog, it can be configured to what column the control a databinding should be performed. For a control in the template that should perform a postback, set the CommandName (and optionally the CommandArgument) property. When the control causes a postback, the event WebPlanner.EventCommand will be triggered with the command name, command type and command argument as parameters.



**Handling template events**



WebPlanner has 5 predefined template control command types:

EventTemplateCommandType.Edit	The control's CommandName is set to "Edit" This assumes the command will put the event into Edit mode.
EventTemplateCommandType.Update	The control's CommandName is set to "Update" This assumes the event was in Edit mode and the command should update the dataview with the edited information
EventTemplateCommandType.CancelEdit	The control's CommandName is set to "CancelEdit" This assumes the event was in Edit mode and this Edit mode should be cancelled. The event will revert to the normal view template without updating the dataview
EventTemplateCommandType.Delete	The control's CommandName is set to "Delete" This assumes the event should be deleted from this control's postback
EventTemplateCommandType.Custom	The control's CommandName is set to an unrecognized name. The event can be in edit or view template and the user should handle the command by checking EventTemplateCommandEventArgs.CommandName

WebPlanner has its built-in handling for the commands : Edit, Update, CancelEdit, Delete but in many cases it is desirable that this is handled in a custom way from WebPlanner.EventCommand. Through bool EventTemplateCommandEventArgs.Handled, it can be set whether the default WebPlanner command handling should be performed or not. When the delegate for WebPlanner.EventCommand fully custom handles the template command, set EventTemplateCommandEventArgs.Handled = true.

Sample template EventCommand delegate

```
protected void WPTPL_EventCommand(object sender,
TMS.WebPlanner.WebPlanner.EventTemplateCommandEventArgs e)
{
    switch (e.CommandType)
    {
        case (TMS.WebPlanner.WebPlanner.EventTemplateCommandType.Update) :

            TextBox tbox = (TextBox)e.Item.FindControl("TextBox1");
            TextBox tbox2 = (TextBox)e.Item.FindControl("TextBox2");
            DropDownList ddl = (DropDownList)e.Item.FindControl("DropDownList1");
            CheckBox cb = (CheckBox)e.Item.FindControl("CheckBox1");

            TMS.WebPlanner.PlannerEvent pe =
(TMS.WebPlanner.PlannerEvent)WPTPL.Items[e.Item.ItemIndex];

            pe.Notes = tbox2.Text;
            pe.Subject = tbox.Text;
            pe.OtherColumns["EventLevel"] = ddl.SelectedValue;
            pe.OtherColumns["Confirmed"] = cb.Checked;

            WPTPL.UpdatePlannerEvent((TMS.WebPlanner.PlannerEvent)WPTPL.Items[e.Item.ItemIndex]);

            WPTPL.StopEdit();
            e.Handled = true;
            break;

        case TMS.WebPlanner.WebPlanner.EventTemplateCommandType.CancelEdit:
            WPTPL.StopEdit();
            e.Handled = true;
```

```
        break;

        case TMS.WebPlanner.WebPlanner.EventTemplateCommandType.Edit:
WPTPL.StartEdit((TMS.WebPlanner.PlannerEvent)WPTPL.Items[e.Item.ItemIndex],
e.Item.ItemIndex);
        e.Handled = true;
        break;

        case TMS.WebPlanner.WebPlanner.EventTemplateCommandType.Delete:
WPTPL.RemoveFromDataSource((TMS.WebPlanner.PlannerEvent)WPTPL.Items[e.Item.ItemIndex]);
        e.Handled = true;
        break;
    }
}
```

In this sample code, the Update command gets the values of the template edit controls, updates the PlannerEvent with the new values and posts these to the database. The Edit command puts the WebPlanner in edit mode while the CancelEdit command puts the WebPlanner out of edit mode. Finally, the Delete command will remove the event from the database.

# Index

---

## A

Active days · 69  
 active zone · 56, 57  
 Active zone · 52  
 ActiveEndTime · 59  
 ActiveEndTime\;DayController · 59  
 ActivePrimary · 52  
 ActiveSecondary · 52  
 ActiveStartTime · 59  
 ActiveStartTime\;DayController · 59  
 AllowDelete · 85  
 AllowOverlap · 85  
 Apply default event colors · 55, 83  
 ASP.NET · 32, 97  
 ASP.NET\;Permissions · 32, 97  
 assemblies · 16  
 assembly · 17  
 Assembly · 18  
 assembly\;design-time · 17  
 assembly\;run-time · 17  
 Assembly\;strong naming · 18  
 AutoDetectCulture · 15

## B

BackColor · 51  
 BackColor\;Header · 51  
 BackColorTo · 51  
 BackColorTo\;Header · 51  
 Background · 85  
 Background event · 70  
 Bands · 52  
 browser · 15  
 browser\;culture · 15

## C

CachedScript property · 58, 84  
 CachedScriptPath property · 58  
 caching · 58, 84  
 Caption · 85, 86, 88  
 Caption\;CaptionStyle · 85  
 Caption\;colors · 86  
 Caption\;Hint · 88  
 CaptionHintStyle · 88  
 captions · 51  
 ChangeQuery event · 93  
 Class hierarchy · 14  
 Client-side JavaScript · 58, 84  
 ColorActive · 56  
 ColorActive\;PositionStyle · 56  
 ColorInactive · 56  
 ColorInactive\;PositionStyle · 56  
 ColorNoSelect · 56  
 ColorNoSelect\;PositionStyle · 56

Controller · 35, 59  
 Controller\;configuring · 35  
 Controller\;connect to WebPlanner · 59  
 convert · 56  
 convert\;cell number to TimeSpan · 56  
 convert\;TimeSpan to cell number · 56  
 culture · 15  
 CurrentDate · 37  
 CurrentDate\;example · 37  
 Custom hint · 88, 90  
 Custom hint\;appearance · 90  
 Custom hint\;duration · 90  
 Custom hint\;pause · 90  
 Custom hint\;position · 90  
 Customization · 18  
 Customize · 20

## D

DataSet · 92, 95  
 DataSet\;autocreate · 95  
 DataSet\;create untyped · 95  
 DataStore · 32, 91, 92  
 DataStore\;configuring · 32  
 DataStore\;connecting to WebPlanner · 91  
 DataStore\;DataSet required · 92  
 DataStore\;EndTimeField · 92  
 DataStore\;field-related properties · 92  
 DataStore\;guidelines · 92  
 DataStore\;introduction · 91  
 DataStore\;KeyField · 92  
 DataStore\;NotesField · 92  
 DataStore\;ResourceField · 92  
 DataStore\;StartTimeField · 92  
 DataStore\;SubjectField · 92  
 DataStores · 91  
 DataStores\;avoiding · 91  
 Date · 37  
 Date\;DayController · 37  
 Day · 52  
 Day header · 69, 79  
 Day mode · 59  
 Day Period mode · 62  
 Day\;inactive · 52  
 DayController · 37, 59  
 DayController\;Date · 37  
 DayController\;Next · 37  
 DayController\;Prev · 37  
 DayHeaderStyle · 79  
 DayIncrement · 59, 62, 65  
 DayIncrement\;DayController · 59  
 DayIncrement\;DayPeriodController · 62  
 DayIncrement\;HalfDayPeriodController · 62  
 DayIncrement\;TimelineController · 65  
 DayMapping · 39, 61  
 DayMapping\;example · 39  
 DayMapping\;MultiDay · 61  
 DayMapping\;MultiDayResource · 61  
 DayMapping\;MultiResource · 39, 61  
 DayMapping\;MultiResourceDay · 61

DayNumberAlignment · 79  
 DayPeriodController · 62  
 Debugging · 18  
 DefaultEvent · 55, 83  
 DeltaX property · 90  
 DeltaY property · 90  
 deploy · 17  
 Detail edit · 43, 72  
 Detail edit\;parameters · 43, 72  
 Detail edit\;refresh parent window · 43, 72  
 DetailEdit.EditTarget · 43, 72  
 DetailEdit.ViewTarget · 43, 72  
 DetailEditParameters · 43, 72  
 DetailUtil · 43, 72  
 DetailUtil\;GetParameters · 43, 72  
 DetailUtil\;RefreshParentWindow · 43, 72  
 Display · 59  
 Display\;TimeUnit · 59

---

## E

EditType · 43, 47, 72, 75  
 EditType\;Custom · 47, 75  
 EditType\;DetailEdit · 43, 72  
 EndDate · 62, 65  
 EndDate\;DayPeriodController · 62  
 EndDate\;HalfDayPeriodController · 62  
 EndDate\;TimelineController · 65  
 EndTime · 59, 85  
 EndTime\;DayController · 59  
 Event · 42, 70, 85, 86  
 Event\;body · 85  
 Event\;caption · 85  
 Event\;colors · 86  
 Event\;create · 42, 70  
 Event\;edit · 70  
 Event\;editPlannerEvent\_create · 42  
 Event\;notes · 85  
 EventCaption · 88  
 EventHints · 88, 90  
 EventHints\;BackColor · 90  
 EventHints\;DeltaX · 90  
 EventHints\;DeltaY · 90  
 EventHints\;Fading · 90  
 EventHints\;Pause · 90  
 EventsRetrieved · 57, 83, 88  
 EventsRetrieved\;setting layers · 57, 83  
 Examples · 15, 28

---

## F

FAQ · 20  
 FixedPosition · 85  
 FixedSize · 85  
 FixedTime · 85  
 Flag · 86  
 ForeColor · 51  
 ForeColor\;Header · 51  
 Frequently Asked Questions · 20

---

## G

GAC · 16, 17  
 Globalization · 15  
 Glyphs · 78, 79  
 Glyphs\;Insert · 79  
 GradientDirection · 51  
 GradientDirection\;Header · 51  
 groups · 51  
 groups\;captions · 51

---

## H

Half Day Period · 49  
 Half Day Period mode · 62  
 Half Day Period\;example · 49  
 HalfDayPeriodController · 62  
 header · 54  
 Header · 25, 51, 54  
 header events · 51  
 Header\;BackColor · 51  
 Header\;BackColorTo · 51  
 header\;captions · 54  
 Header\;captions · 51  
 Header\;ForeColor · 51  
 Header\;GradientDirection · 51  
 Header\;Position · 51  
 Header\;UpdateCaptions · 54  
 HeaderClick event · 51  
 Help · 15  
 Hierarchy · 14  
 Hint · 87, 88, 90  
 Hint\;appearance · 90  
 Hint\;backcolor · 90  
 Hint\;duration · 90  
 Hint\;position · 90  
 How To · 28

---

## I

IController · 15  
 IDataStore · 15  
 In Place edit · 42, 70  
 Inactive days · 52, 69  
 Inactive zone · 52  
 InactiveDays · 79  
 InactiveDaysStyle · 79  
 InactivePrimary · 52  
 InactiveSecondary · 52  
 Installation · 15, 16  
 Installation\;Visual Studio.NET · 16  
 interfaces · 15  
 IPlanner · 15  
 Items collection · 55, 83

---

## J

JavaScript · 58, 84  
 JavaScript\;cache · 58, 84

---

**K**

Keyboard support · 57, 84  
 Keys · 57, 84  
 Keys\;arrow · 57, 84  
 Keys\;Delete · 57, 84  
 Keys\;Insert · 57, 84

---

**L**

Layers · 57, 83  
 License agreement · 19

---

**M**

Manual · 19  
 Manual\;Organization · 19  
 mode · 59, 62, 63, 64, 65, 66  
 mode\;configure · 59  
 mode\;Day · 59  
 mode\;Day Period · 62  
 mode\;Half Day Period · 62  
 mode\;Month · 63  
 mode\;MultiMonth · 64  
 mode\;Timeline · 65  
 mode\;Week · 66  
 Month · 63, 66, 70, 79  
 Month mode · 63  
 Month\;MonthPeriodController · 63  
 Month\;WeekController · 66  
 MonthNames · 78  
 MonthPeriodController · 63  
 MonthPlanner · 69, 70, 72, 75, 78, 79, 81, 83, 84, 85, 88, 90  
 MonthPlanner\;active days · 69  
 MonthPlanner\;CachedScript · 84  
 MonthPlanner\;CaptionHintStyle · 88  
 MonthPlanner\;CellsSelected event · 75  
 MonthPlanner\;create events · 70  
 MonthPlanner\;custom edit · 72  
 MonthPlanner\;day header · 69  
 MonthPlanner\;DayHeaderStyle · 79  
 MonthPlanner\;DayNames · 79  
 MonthPlanner\;DayNumberAlignment · 79  
 MonthPlanner\;DefaultEvent · 83, 85  
 MonthPlanner\;detail edit · 72  
 MonthPlanner\;edit events · 70  
 MonthPlanner\;EditType · 70, 72, 75  
 MonthPlanner\;EventCaption · 88  
 MonthPlanner\;EventCustomEdit event · 75  
 MonthPlanner\;EventHints · 88, 90  
 MonthPlanner\;EventRetrieved · 85  
 MonthPlanner\;Events · 83  
 MonthPlanner\;EventsRetrieved · 83, 85, 88  
 MonthPlanner\;geography · 69  
 MonthPlanner\;Glyphs · 78, 79  
 MonthPlanner\;in place edit · 70  
 MonthPlanner\;inactive days · 69  
 MonthPlanner\;InactiveDays · 79  
 MonthPlanner\;InactiveDaysStyle · 79  
 MonthPlanner\;Items · 85  
 MonthPlanner\;keyboard · 84  
 MonthPlanner\;Layer · 83

MonthPlanner\;Month · 70, 79  
 MonthPlanner\;MonthNames · 78  
 MonthPlanner\;other month days · 69  
 MonthPlanner\;OtherMonthDayStyle · 81  
 MonthPlanner\;PeriodStyles · 81  
 MonthPlanner\;popup edit · 70  
 MonthPlanner\;SelectionMode · 72  
 MonthPlanner\;ShowDaysInOtherMonth · 81  
 MonthPlanner\;ShowYearInTitle · 78  
 MonthPlanner\;StartDay · 79  
 MonthPlanner\;title · 69  
 MonthPlanner\;Title · 78  
 MonthPlanner\;TitleStyle · 79  
 MonthPlanner\;today · 69  
 MonthPlanner\;Year · 70, 79  
 MonthPlannerUtil · 72  
 MonthPlannerUtil\;CellToDateTime · 72  
 MultiDay · 61  
 MultiDayResource · 61  
 MultiMonth mode · 64  
 MultiMonthController · 64  
 MultiResource · 61  
 MultiResourceDay · 61

---

**N**

New features · 11  
 Next · 37  
 Next\;DayController · 37  
 Next\;example · 37  
 Notes · 85, 86  
 Notes\;colors · 86  
 NumberOfDays · 59, 62, 65  
 NumberOfDays\;DayController · 59  
 NumberOfDays\;DayPeriodController · 62  
 NumberOfDays\;HalfDayPeriodController · 62  
 NumberOfDays\;TimelineController · 65  
 NumberOfMonths · 64  
 NumberOfMonths\;MultiMonthController · 64

---

**O**

OleDbConnection · 93  
 OleDbDataStore · 92, 93  
 OleDbDataStore\;ChangeQuery · 93  
 OleDbDataStore\;configuring · 92, 93  
 Other month days · 69  
 OtherMonthDayStyle · 81  
 overlap · 25  
 Overlap column · 25  
 overlapping · 25  
 Overview · 11

---

**P**

Period style · 81  
 PeriodStyle · 81  
 PeriodStyle\;BackColor · 81  
 PeriodStyle\;EndDate · 81  
 PeriodStyle\;ReadOnly · 81  
 PeriodStyle\;StartDate · 81

Permissions · 32, 97  
 Permissions\;ASP.NET · 32, 97  
 PlannerDisplay · 49  
 PlannerDisplay\;RowHeight · 49  
 PlannerEndTime · 85  
 PlannerEvent · 20, 22, 25, 42, 55, 57, 70, 83, 85, 86, 87, 88, 91  
 PlannerEvent\;AllowDelete · 85  
 PlannerEvent\;AllowOverlap · 85  
 PlannerEvent\;BackColor · 86  
 PlannerEvent\;Background · 70, 85  
 PlannerEvent\;Caption · 85, 86  
 PlannerEvent\;Caption hint · 88  
 PlannerEvent\;colors · 86  
 PlannerEvent\;create · 70  
 PlannerEvent\;Custom hint · 88  
 PlannerEvent\;Customize · 20  
 PlannerEvent\;default · 55, 83  
 PlannerEvent\;edit · 42, 70  
 PlannerEvent\;EndTime · 85  
 PlannerEvent\;FixedPosition · 85  
 PlannerEvent\;FixedSize · 85  
 PlannerEvent\;FixedTime · 85  
 PlannerEvent\;FlagColor · 86  
 PlannerEvent\;Flagged · 86  
 PlannerEvent\;Font · 86  
 PlannerEvent\;ForeColor · 86  
 PlannerEvent\;Hide · 22  
 PlannerEvent\;Hint · 87, 88  
 PlannerEvent\;Layer · 57, 83  
 PlannerEvent\;managed by application · 91  
 PlannerEvent\;Managing without data store · 22  
 PlannerEvent\;Notes · 85  
 PlannerEvent\;PlannerEndTime · 85  
 PlannerEvent\;PlannerStartTime · 85  
 PlannerEvent\;Prevent move · 22  
 PlannerEvent\;Prevent resize · 22  
 PlannerEvent\;ReadOnly · 22, 55, 83, 85  
 PlannerEvent\;SelectionColor · 86  
 PlannerEvent\;SelectionFontColor · 86  
 PlannerEvent\;StartTime · 85  
 PlannerEvent\;Tool tip · 87  
 PlannerEvent\;Update times · 22  
 PlannerEvent\;Visible · 22  
 PlannerSideBar · 49  
 PlannerSideBar\;BackColor · 49  
 PlannerSideBar\;BackColorTo · 49  
 PlannerSideBar\;occupied · 49  
 PlannerSideBar\;OccupiedColor · 49  
 PlannerSideBar\;OccupiedColorTo · 49  
 PlannerSideBar\;Position · 49  
 PlannerSideBar\;ShowOccupied · 49  
 PlannerSideBar\;TextInterval · 49  
 PlannerSideBar\;visible · 49  
 PlannerSideBar\;Width · 49  
 PlannerStartTime · 85  
 Popup edit · 42, 70  
 position · 55, 56  
 Position · 25, 54, 90  
 position style · 56  
 position style\;colors · 56  
 position\;associated resource · 55  
 Position\;Custom hint · 90  
 position\;style · 56  
 positions · 51  
 Positions collection · 54

positions\;captions · 51  
 PositionStyle · 22, 56, 57  
 PositionStyle\;read only · 57  
 PositionStyle\;ReadOnly · 22  
 PositionStyles · 56  
 Prev · 37  
 Prev\;DayController · 37  
 Prev\;example · 37

---

## R

Read only · 55, 57, 83  
 ReadOnly · 85  
 Recompiling · 18  
 Refresh parent window · 43, 72  
 Register directive · 14  
 resource · 55  
 Resource · 25  
 Resources · 39  
 Resources collection · 55  
 Resources\;Day mode · 39  
 Resources\;example · 39  
 Read only · 81

---

## S

Security · 22  
 SelectionMode · 43, 47, 72, 75  
 SelectionMode.SingleSelectAutoCreate · 43, 72  
 SelectionMode\;MultiSelect · 47, 75  
 SelectionMode\;MultiSelectAutoCreate · 43, 72  
 SelectionMode\;SingleSelect · 47, 75  
 ShowDaysInOtherMonth · 81  
 ShowYearInTitle · 78  
 SideBar · 25  
 Source code · 18  
 Source code\;customization · 18  
 Special events · 86  
 SqlClientDataStore · 92, 93  
 SqlClientDataStore\;ChangeQuery · 93  
 SqlClientDataStore\;configuring · 92, 93  
 SqlConnection · 93  
 StartDate · 62, 65  
 StartDate\;DayPeriodController · 62  
 StartDate\;HalfDayPeriodController · 62  
 StartDate\;TimelineController · 65  
 StartDay · 79  
 StartMonth · 64  
 StartMonth\;MultiMonthController · 64  
 StartTime · 59, 85  
 StartTime\;DayController · 59  
 Status · 86  
 Strong naming · 18  
 System requirements · 14

---

## T

Tag · 86  
 Timeline mode · 65  
 TimelineController · 65  
 TimeUnit · 59

Title · 69, 78  
 TitleStyle · 79  
 Tool tip · 87  
 TopLeftCell · 51  
 TrackBar · 25  
 Trogdor · 88  
 Tutorial · 28

---

## U

Upgrade · 14

---

## W

WebPlanner · 14, 22, 42, 43, 47, 51, 54, 55, 56, 57, 58, 59, 85, 88, 90, 91  
 WebPlanner\;CachedScript · 58  
 WebPlanner\;CachedScriptPath · 58  
 WebPlanner\;CaptionHintStyle · 88  
 WebPlanner\;CellsSelected event · 47  
 WebPlanner\;configure mode · 59  
 WebPlanner\;connect Controller · 59  
 WebPlanner\;connecting to DataStore · 91  
 WebPlanner\;create events · 42  
 WebPlanner\;custom edit · 43  
 WebPlanner\;DefaultEvent · 55, 85  
 WebPlanner\;detail edit · 43  
 WebPlanner\;edit events · 42  
 WebPlanner\;EditType · 42, 43, 47  
 WebPlanner\;EventCaption · 88  
 WebPlanner\;EventCustomEdit event · 47  
 WebPlanner\;EventHints · 88, 90  
 WebPlanner\;EventRetrieved · 85  
 WebPlanner\;Events · 55  
 WebPlanner\;EventsRetrieved · 57, 85, 88  
 WebPlanner\;HeaderClick · 51  
 WebPlanner\;in place edit · 42  
 WebPlanner\;Items · 85  
 WebPlanner\;keyboard · 57  
 WebPlanner\;Layer · 57  
 WebPlanner\;popup edit · 42  
 WebPlanner\;Positions · 54  
 WebPlanner\;PositionStyles · 56  
 WebPlanner\;ReadOnly · 22

WebPlanner\;Resources · 55  
 WebPlanner\;Security · 22  
 WebPlanner\;SelectionMode · 43  
 WebPlanner\;upgrade · 14  
 WebPlannerUtil · 43  
 WebPlannerUtil\;CellToDateTime · 43  
 Week mode · 66  
 WeekController · 66  
 Weeks · 66  
 Weeks\;WeekController · 66

---

## X

XML · 97  
 XML\;Load · 97  
 XML\;Save · 97  
 XmlDataStore · 32, 92, 94, 95, 97  
 XmlDataStore\; · 94  
 XmlDataStore\;configuring · 92, 94  
 XmlDataStore\;DataSet · 97  
 XmlDataStore\;example · 32  
 XmlDataStore\;fields · 95  
 XmlDataStore\;persistence · 97  
 XmlDataStore\;read XML · 97  
 XmlDataStore\;write XML · 97

---

## Y

Year · 63, 64, 66, 70, 79  
 Year\;MonthPeriodController · 63  
 Year\;MultiMonthController · 64  
 Year\;WeekController · 66

---

## Z

zone · 52, 56, 57  
 zone\;active · 52, 56, 57  
 zone\;inactive · 52  
 zone\;no events added · 57  
 zone\;no selection allowed · 57  
 ZoneAlarm · 14