TMS Silverlight - WPF Planner
DEVELOPERS GUIDE

Aug 2010

Web: http://www.tmssoftware.com
Email : info@tmssoftware.com

Index

TMS Silverlight - WPF Planner use

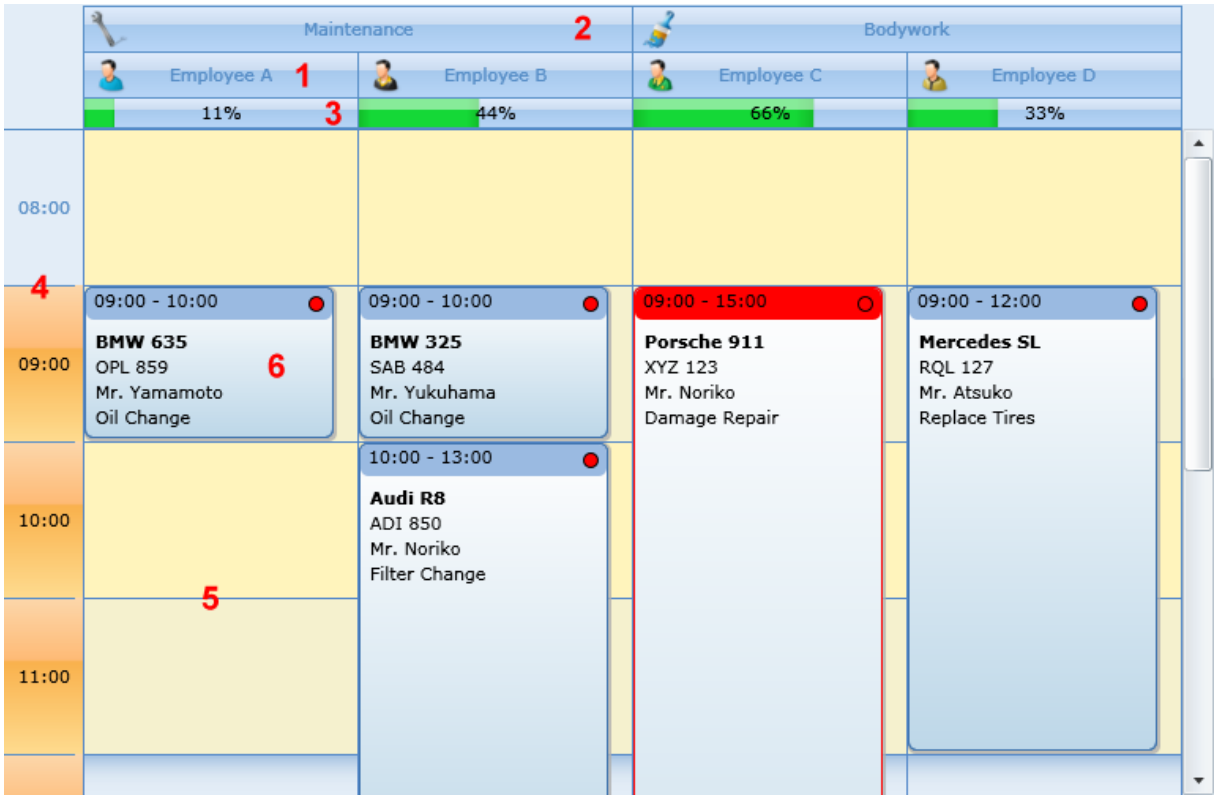
The TMS Silverlight - WPF Planner component is designed to be used in the broadest types of planning and scheduling type of applications. This can range from the typical single person PIM application to schedulers of activities for a group of persons, time planning for resources such as hotel rooms, car rental, university courses and so much more. As such, the TMS Silverlight - WPF Planner is a very highly configurable component to suite all these various types of applications. The underlying framework of the Planner therefore has an open interface towards the coupling to time or resources. Standard modes include day time view, multi-day view, month view, multi-resource view while at the same time custom modes allow to view any type of timescale. In all these time modes, there is also optional support for recurrency.

- TMS Silverlight Planner was developed using Microsoft Visual Studio 2008-2010 using C# and supports Microsoft Silverlight 2, 3 and 4.
- TMS WPF Planner was developed using Microsoft Visual Studio 2010 using C# and supports Microsoft WPF 4.0.

TMS Silverlight - WPF Planner organisation

The visual organisation of TMS Silverlight - WPF Planner

TMS Silverlight - WPF Planner is a component with following main visual elements:



1) **Header**: Describes the content of each column. Optional.

2) **HeaderGroup**: Group one or more header items together. Optional.

3) **Completion**: Zone where completion of a resource can be displayed by a progress bar. Optional

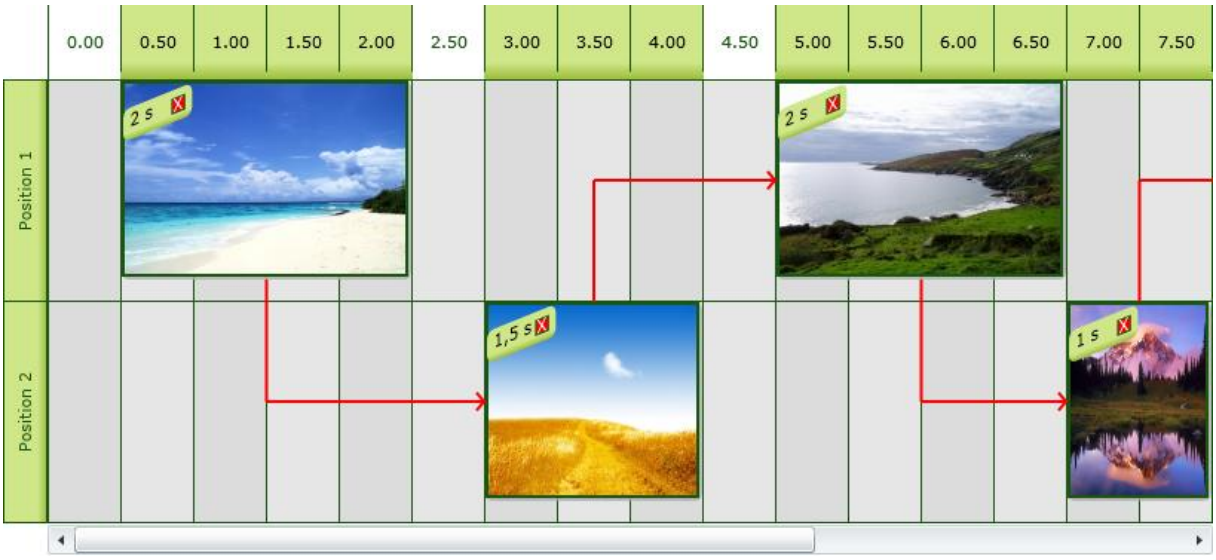
4) **TimeAxis**: Settings are controlled through the Planner.TimeAxis* properties. The TimeAxis can be visible or not. It can be at the left side, right side, left and right side, or top side of the grid. In addition, it can also optionally be repeated between columns in the grid.

5) **Grid**: Various appearance settings are controlled through the Planner properties: ActiveRowBrush, InactiveRowBrush, …

6) **PlannerItem**: The default PlannerItem has a caption, a notes section, a delete button and optionally also a recurrency indicator.

Display Modes

Silverlight - WPF Planner supports a vertical view (as shown above) as well as a horizontal view (as shown below). In the horizontal view, all elements of the Planner are simply rotated 90 degrees. This means that the TimeAxis here at left is displayed on top and the header is displayed at the left side in horizontal mode. The position coordinates of PlannerItem objects along vertical and horizontal axis in vertical mode become the coordinates along horizontal and vertical axis respectively in horizontal mode.
The Planner component can be easily switched from vertical view to horizontal view by setting the Planner.Mode property to SelectionMode.Horizontal.



Keyboard Support

In general the user interacts with the Planner using the mouse cursor, but the planner can also be controlled by using the keyboard:

- Use the arrow-keys (up, down, left right) to navigate through the Planner's grid. (holding the Shift-key allows to select multiple cells)
- Use the insert key to add a new item, after selecting one or more cells.
- Use the delete key to remove the currently selected item.
- Use the F2 or enter key to start editing the currently selected item.

Note: The Planner must have focus to receive the keyboard input.

The programmatic organisation of the TMS Silverlight - WPF Planner

Properties Overview

- ActiveRowBrush
  - Brush, the brush of an active row. An active row means a row in the active time zone of a Planner timespan. The Planner can indicate inactive (non working hours for example) rows in a different color from active rows (working hours for example)

- ActiveRowSecondaryBrush
  - Brush, the secondary brush of an active row. Set this property to a different brush then ActiveRowBrush to achieve a banding effect in the active timespan of the Planner.

- AutoCreateOnSelect
  - Boolean, indicates if an item is automatically created when one or more cells of the Planner have been selected. The creation of a new PlannerItem happens in the mouse up if this option is set.

- AutoEditOnCreate
  - Boolean, indicates if an item is automatically set to edit mode when it's inserted on the Planner.

- AutoInsDel
  - Boolean, indicates if items can be inserted with the insert key or by selecting one or more cells with the mouse and if items can be deleted with the delete key or by clicking the deletebutton in the item's caption.

- ActiveStartTime
  - DateTime, sets at what time the active period of the Planner's TimeAxis starts. Cells before this time will get the InActiveBrush/InActiveSecondaryBrush as their background; cells after this time will get the ActiveBrush/ActiveSecondaryBrush as their background.

- ActiveEndTime
  - DateTime, sets at what time the active period of the Planner's TimeAxis ends. Cells before this time will get the ActiveBrush/ActiveSecondaryBrush as their background; cells after this time will get the InActiveBrush/InActiveSecondaryBrush as their background.

- BorderBrush
  - Brush, that controls the appearance of the cell borders.

- CaptionContent
  - DataTemplate, enables using a templated caption.

- ColCount
  - Integer, the current number of columns displayed in the Planner.

- ColWidth
  - Double, the width of a column.

- CompletionPosition
  - Position, the position of the Completion header. The completion indicates the percentage of time used by PlannerItems in the viewed active or full timespan of the Planner. The completion can be displayed on the first header, the last header or both. When Mode is set to Vertical first is on top and last is at the bottom, when

Mode is set to Horizontal first is at the left side and last is at the right side. Setting the CompletionPosition to None hides the completion header.

- CompletionRange
  - Range, indicates how the completion percentage is calculated. When set to Range.Active completion is calculated from ActiveStartTime to ActiveEndTime, when set to Range.Full completion is calculated from StartTime to EndTime.

- ConfirmDeleteText
  - String, the text displayed in the confirmation dialog when deleting a PlannerItem.

- ConfirmInsertText
  - String, the text displayed in the confirmation dialog when inserting a PlannerItem.

- Databinding
  - PlannerDatabinding, assign your custom PlannerDatabinding class to perform databinding.

- DataContext (Only available in the WPF Planner, see ItemsSource when using the Silverlight Planner)
  - Object, used to generate the PlannerItems list when using a databound Planner.

- Day
  - DateTime, sets the day from where the Planner viewed timespan starts.

- DayMapping
  - DayMapType, indicates the way days and resources are displayed.
    - MultiDay: Every column represents one day
    - MultiResource: Every column represents a resource
    - MultiDayResource: Resource columns are repeated for every day
    - MultiResourceDay: Day columns are repeated for every resource

- DetailEditHeight
  - Double, the height of a PlannerItem in edit mode when EditMode is set to Detail.

- DetailEditWidth
  - Double, the width of a PlannerItem in edit mode when EditMode is set to Detail.

- EditIndex
  - Integer, the index of the PlannerItem that is being edited.

- EditMode
  - EditModes, the way a PlannerItem can be edited.
    - Detail: The event is resized to the size defined in the DetailEditHeight and DetalEditWidth properties and the EditingDataTemplate content is displayed while the PlannerItem is in edit mode.
    - Inplace: default behaviour, the item's text can be edited like a TextBox inplace in the Notes area of the item.
    - PopupDetail: a custom form is displayed above the item that is being edited.

- EndTime
  - DateTime, the endtime of the Planner.

- FieldFirstEndTime

  - String, the field in the database that contains the endtime of the first occurrence of a recurrent item.

- FieldFirstStartTime
  - String, the field in the database that contains the starttime of the first occurrence of a recurrent item.

- FieldEndTime
  - String, the field in the database that contains the endtime of an item.

- FieldKey
  - String, the field in the database that contains the ID of an item.

- FieldNotes
  - String, the field in the database that contains the description text of an item.

- FieldRecurrency
  - String, the field in the database that contains the recurrency rule of an item.

- FieldReource
  - String, the field in the database that contains the resource index of an item.

- FieldStartTime
  - String, the field in the database that contains the starttime of an item.

- FieldResource
  - String, the field in the database that contains the resource index of an item.

- FieldSubject
  - String, the field in the database that contains the caption/subject text of an item.

- Gap
  - Double, the size of the gap between an item and the cell border. The gap can be used to add multiple (overlapping) items to the same cell.

- Groups
  - PlannerGroupList, a list of columnheaders that group one or more columns.

- EventsInHeaderBackground
  - Brush, the background brush of the region where items are displayed in the header.

- InactiveRowBrush
  - Brush, the brush of an inactive row, i.e. row in the inactive timespan of the Planner.

- InActiveRowSecondaryBrush
  - Brush, the secondary brush of an inactive row. Set this property to a different brush then InActiveRowBrush to achieve a color banding effect.

- Interval
  - TimeSpan, the duration of a single TimeAxis cell.

- Items
  - PlannerItemList, a list of PlannerItems. Create a new PlannerItem and add it to the Items list to display it on the Planner. When databinding is performed, this list is automatically populated by the dataset.

- ItemsSource (Only available in the Silverlight Planner, see DataContext when using the WPF Planner)
  - IENumerable, a collection used to generate the PlannerItems list when using a databound Planner.

- ItemStyle
  - Style, the default style template for a PlannerItem.

- ItemContent
  - DataTemplate, enables using a templated item content.

- ItemEditingContent
  - DataTemplate enables using a templated item content when the item is in edit mode.

- Layer
  - Int64, The layer displayed in the Planner. The layers use a binary compare between the Planner's Layer and the PlannerItem's Layer to determine which items are displayed. The value 0 for Layer means all PlannerItems are displayed. When the Layer is set to 1, all PlannerItems where bit 0 is set for the PlannerItem Layer will be displayed in the Planner. When the Layer is set to 4, all PlannerItems where bit 2 is set for the PlannerItem Layer will be displayed in the Planner. To show items belonging to layer 1 and 2, set Planner Layer to 3 etc...

- MultiSelect
  - Boolean, indicates if multiple PlannerItems can be selected. Select multiple items by holding down the Ctrl key and clicking an item.

- MultiCellSelect
  - Boolean, indicates if multiple cells can be selected.

- MultiUser
  - Boolean, indicates if multiple users can insert/update/delete items on the same view. If set to true the Planner calls the Databinding.IsDirty method every 5 seconds.

- Mode
  - SelectionMode, indicates if the Planner is displayed in Horizontal or Vertical Mode.

- NumberOfDays
  - Integer, indicates how many days are displayed on the Planner when DayMapType is set to MultiDay, MultiDayResource, MultiResourceDay. A column (in Vertical Mode or a row in Horizontal Mode) is automatically generated for every day.

- PeriodEndDate
  - DateTime, contains the end DateTime of the Planner when NumberOfDays is > 1

- PeriodStartDate
  - DateTime, contains the start DateTime of the Planner when NumberOfDays is > 1

- Positions
  - PlannerPositionList, a list of planner positions. A column (in Vertical Mode or a row in Horizontal Mode) is generated for each PlannerPosition in the list when the Planner DayMapType is set to MultiResource, MultiDayResource or MultiResourceDay.

- RecurrencyGlyph

  - String, the path to the image that is used to indicate that an item is recurrent.

- RowHeight
  - Double, the height of a row

- RowCount
  - Integer, the current number of rows displayed in the Planner.

- ScrollPositionHorizontal
  - Double, the scroll position of the horizontal scroll bar when the Planner is first displayed.

- ScrollPositionVertical
  - Double, the scroll position of the vertical scroll bar when the Planner is first displayed.

- SelectedIndex
  - Integer, the index of the currently selected PlannerItem.

- SelectionBrush
  - Brush, the brush of selected Planner cells.

- SelectionEnd
  - Integer, the index of the last row (in Vertical Mode or column in Horizontal Mode) that is currently selected.

- SelectionPosition
  - Integer, the index of the column (in Vertical Mode, or row in Horizontal Mode) that currently has one or more selected cells.

- SelectionStart
  - Integer, the index of the first row (in Vertical Mode, or column in Horizontal Mode) that is currently selected.

- StartTime
  - DateTime, the start time of the Planner.

- ToolTip
  - Object, the object displayed in the Planner tooltip.

Header Properties

- HeaderBackBrush
  - Brush, the background of the planner header.

- HeaderBorderBrush
  - Brush, the border of the planner header.

- HeaderBrush
  - Brush, the brush of the text in the planner header.

- HeaderDisplayFormat
  - String, holds the format string for DateTime values that are displayed in the Header. Applied when the DayMapping property is set to DayMapType.MultiDay, DayMapType.MultiDayResource or DayMapType.MultiResourceDay. Default format is

"dddd dd MMMM".

- HeaderPosition
  - Position, the position of the Header. The Header can be set to positon first, last or both. When Mode is set to Vertical, first is on top and last is at the bottom, when Mode is set to Horizontal, first is at the left side and last is at the right side. Setting the HeaderPosition to None hides the header.

- HeaderRotate
  - Boolean, decides if the Header text is displayed horizontally or vertically when the Planner Mode is set to Horizontal.

- HeaderSize
  - Double, the height of the header when in Vertical Mode or the width of the header when in Horizontal Mode.

TimeAxis properties

- TimeAxisBrush
  - Brush, the text brush of the TimeAxis cells.

- TimeAxisBackBrush
  - Brush, the background brush of the TimeAxis cells.

- TimeAxisBorderBrush
  - Brush, the border brush of the TimeAxis cells.

- TimeAxisCurrentBrush
  - Brush, the text brush of the TimeAxis cell that displays the current time, i.e. the local system time on the client machine.

- TimeAxisCurrentBackBrush
  - Brush, the background brush of the TimeAxis cell that displays the current time.

- DisplayFormat
  - String, holds the format string for DateTime values that are displayed in the TimeAxis. Default format is "HH:mm".

- OccupiedBrush
  - Brush, the text brush for a TimeAxis cell of a Planner row that contains one or more PlannerItems.

- OccupiedBackBrush
  - Brush, the background brush for a TimeAxis cell of a Planner row that contains one or more PlannerItems.

- TimeAxisPosition
  - Position, the position of the TimeAxis. The TimeAxis can be displayed first, last or both. When Mode is set to Vertical, first is on top and last is at the bottom, when Mode is set to Horizontal, first is at the left side and last is at the right side. Setting the TimeAxisPosition to None hides the TimeAxis.

- TimeAxisRepeat
  - Boolean, indicates if the TimeAxis is displayed next to each column (in Vertical Mode or each row in Horizontal Mode).

- TimeAxisRotate
  - Boolean, indicates if the TimeAxis text is displayed horizontally or vertically when the Planner Mode is set to Vertical.

- ShowCurrent
  - Boolean, indicates if the TimeAxisCurrentBrush and TimeAxisCurrentBackBrush properties are used.

- TimeAxisSize
  - Double, sets the width of the TimeAxis in Vertical Mode or the height of the TimeAxis in Horizontal Mode.

PlannerItem Properties

- AllowOverlap
  - Boolean, indicates if a PlannerItem can be overlapped. When overlapping is allowed (also often called conflicting items) multiple Items can be allocated on the same time slot, ie. Planner cell or cells. Overlapping or conflicting PlannerItems are automatically resized to fit in the Planner column (or row).

- BackgroundItem
  - Boolean, indicates if the PlannerItem is displayed behind other Items. A BackgroundItem is a PlannerItem that is always non-conflicting but can be overlapped. Overlapping PlannerItems are just displayed on top of background PlannerItems.

- CaptionBrush
  - Brush, the caption brush of a PlannerItem.

- CaptionText
  - String, the text displayed in the caption of a PlannerItem.

- CaptionType
  - CaptionTypes, the type of information that is displayed in the caption.
    - Text: the value of the CaptionText property is displayed.
    - Time: the StartTime and EndTime of the Item is displayed.
    - TimeText: both the value of the CaptionText and the StartTime and EndTime of the Item are displayed.

- DBItem
  - Boolean, sets if the Item was retrieved from databinding (when true). This value is automatically set to true for Items added trough databinding or to false for programmatically added Items. See the Planner methods ClearDBItems and ClearNonDBItems.

- DeleteButtonToolTip
  - Object, the object that is displayed in the tooltip of the Item's delete button.

- EndPos
  - Integer, index of the last row the PlannerItem is displayed on.

- EndTime
  - DateTime, the endtime of the PlannerItem.

- FixedResource

  - Boolean, sets if the PlannerItem can be moved to another Resource (when false).

- FixedDuration
  - Boolean, sets if the PlannerItem can be resized (when false).

- FixedStartTime
  - Boolean, sets if the starttime of a PlannerItem can be changed (when false).

- FixedEndTime
  - Boolean, sets if the endtime of a PlannerItem can be changed (when false).

- ForeColor
  - Brush, the brush of the Item's Text.

- InHeader
  - Boolean, sets if the PlannerItem is displayed in the Planner Header. This property is set automatically to true when the Item's StartTime is earlier than the Planner's StartTime and the Item's EndTime is later than the Planner's EndTime.

- IsParent
  - Boolean, indicates if the item is the Parent recurrent Item. This is automatically set to true when this is the first occurrence of a recurrent item.

- Key
  - String, contains the unique ID of the Item.

- Layer
  - Int64, the layer of the Item. The Item is displayed on the Planner if the Layer value is contained in the Planner.Layer property. See also the Planner.Layer property.

- LinkBrush
  - Brush, the brush of the PlannerItem's Link.

- LinkedItem
  - PlannerItem, the PlannerItem that is linked with this PlannerItem. If a PlannerItem has a LinkedItem defined, an arrow is displayed between the two Items.

- LinkType
  - LinkTypes, the behaviour of a linked Item.
    - StartAndEnd: when the Start- or EndTime of the Item changes, the Start- and EndTime of the linked Item also changes.
    - StartStart: when the StartTime of the Item changes, the StartTime of the linked item also changes.
    - StartEnd: when the StartTime of the Item changes, the EndTime of the linked Item also changes.
    - EndStart: when the EndTime of the Item changes, the StartTime of the linked Item also changes.
    - EndEnd: when the EndTime of the Item changes, the EndTime of the linked item also changes.

- Parent
  - PlannerItem, the parent Item of this Item. Contains the Item that is the first occurrence of this item (for recurrent items).

- ReadOnly
  - Boolean, sets if the PlannerItem is read-only.

- Recurrency
  - String, the PlannerItem's recurrency formula.
- RecurrencyStartTime
  - DateTime, the StartTime of the Item's first occurrence (for recurrent items).
- RecurrencyEndTime
  - DateTime, the EndTime of the Item's last occurrence (for recurrent items).
- RecurrencyStartTimeFirst
  - DateTime, the StartTime of the Item's first occurrence (for recurrent items).
- RecurrencyEndTimeFirst
  - DateTime, the EndTime of the Item's first occurrence (for recurrent items).
- ResPos
  - Integer, the index of the resource this Item belongs to.
- ResourceID
  - String, the ID of the resource this Item belongs to.
- Selected
  - Boolean, indicates or sets if the PlannerItem is selected.
- ShowDeleteButton
  - Boolean, sets if the PlannerItem's delete button is displayed.
- ShowTrackBar
  - Boolean, sets if the PlannerItem's trackbar is displayed.
- StartPos
  - Integer, the index of the first row the PlannerItem is displayed on.
- StartTime
  - DateTime, the starttime of the PlannerItem.
- Stroke
  - Brush, the border brush of the PlannerItem.
- StrokeThickness
  - Thickness, the borderwidth of the PlannerItem.
- Text
  - String, the description/notes text of the Item.
- ToolTip
  - Object, the object that is displayed in the tooltip of the Item.
- TrackBarBackBrush
  - Brush, the background brush of the Item's trackbar.
- TrackBarBrush
  - Brush, the brush of the Item's trackbar.
- TrackBarStroke
  - Brush, the border brush of the Item.

- Visible
  - Boolean, indicates if the Item is displayed on the Planner.

Events Overview

- CellClick
  - Fires when a mouseclick occurs on a cell.
- GetCellBackground
  - Fires for every cell that is created. Set the background brush using the Background parameter.
- GetCellText
  - Fires for every cell that is created. Set the text to appear in the cell using the Text parameter.
- GetTimeAxisText
  - Fires for every TimeAxis cell that is created. Set the text to appear in the cell using the Text parameter.
- GetTimeAxisTime
  - Fires for every TimeAxis cell that is created. Set the time value to appear in the cell using the TimeStamp parameter.
- ItemCaptionClick
  - Fires when a mouseclick occurs on the Item's caption.
- ItemRecurrencyClick
  - Fires when a mouseclick occurs on the Item's recurrency symbol (for recurrent items).
- ItemInserted
  - Fires when an Item is added to the Item List. Set the Allow parameter to AllowType.AllowWithConfirmation to allow inserting the item after confirmation; AllowType.AllowWithoutConfirmation to allow inserting the item without confirmation; AllowType.NotAllowed if the Item may not be inserted.
- ItemDeleted
  - Fires when an Item is removed from the Item List. Set the Allow parameter to AllowType.AllowWithConfirmation to allow deleting the item after confirmation; AllowType.AllowWithoutConfirmation to allow deleting the item without confirmation; AllowType.NotAllowed if the Item may not be deleted.
- ItemSelected
  - Fires when an Item is selected (when a mouseclick occurs on the Item).
- ItemMoved
  - Fires when an Item is moved.
- ItemSized
  - Fires when an Item is resized.
- ItemEditing
  - Fires when an Item is set to editmode.

- ItemEdited
  - Fires when an Item is done editing.

- ItemRetrieved
  - Fires each time an Item has been added to the Planner's Items collection (when using a databound Planner).

- ItemsRetrieved
  - Fires after all items have been added to the Planner's Items collection (when using a databound Planner).

- Refreshed
  - Fires when the Planner is refreshed.

- SelectionChanged
  - Fires when the cells selection has changed.

- SelectionChanging
  - Fires every time the cells selection changes, when selecting cells using the mouse.

Methods Overview

- BeginUpdate
  - It's recommended to always use this method before changing multiple Planner or PlannerItem properties to avoid multiple refreshes. Calling this method tells the Planner it's in update mode and it won't refresh each time a property has changed.
    - Call the EndUpdate method after the Planner's properties have been changed to do a Planner refresh using the new property values.
    - Call the StopUpdate method after the Planner's properties have been changed to end update mode and you don't want the Planner to update immediately.

- ClearDBItems
  - Removes all items form the Item List that where added trough databinding (where Item.DBItem is set to true).

- ClearNonDBItems
  - Removes all items from the Item List that where added programmatically (where Item.DBItem is set to false).

- DisableUI
  - Disable all user interaction with the Planner.

- EnableUI
  - Enable all user interaction with the Planner.

- EndUpdate
  - It's recommended to use this method after changing multiple Planner properties to avoid multiple refreshes. Calling this method tells the Planner it's no longer in update mode and refreshes the Planner using the new property values.
    - Call the BeginUpdate method before changing the properties.

- IllegalyOverlaps(PlannerItem)
  - Returns true if an overlapping Item was found, returns false if no overlapping items where found.
    Use this method to check if an Item overlaps with any other Item, that has

OverlapAllowed set to False, currently displayed on the Planner.

- Refresh
  - Refresh the Planner. The Planner control is completely redrawn. The Planner is automatically refreshed if any of the properties are changed. When changing multiple properties at the same time, it's recommended to use the BeginUpdate/EndUpdate/StopUpdate methods.

- StopUpdate
  - Use this method after changing multiple Planner properties and you don't want the Planner to refresh immediately. Calling this method tells the Planner it's no longer in update mode.
    - Call the BeginUpdate method before changing the properties.

Display Types

A) **Default**

By default the Planner will display the current day, from 0:00 hours to 24:00 hours with an interval of 1 hour. These settings can be changed using the StartTime, EndTime and Interval properties.

B) **MultiResource**

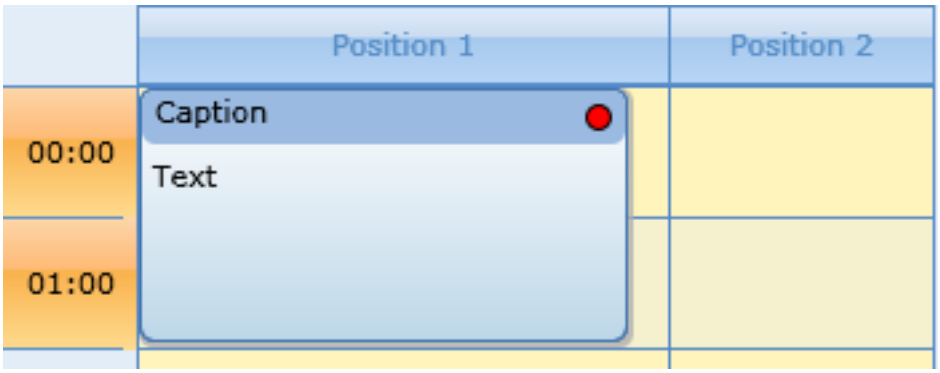
Add a PlannerPosition to the Planner's Position collection for each resource that needs to be displayed. Via the Name property, the text displayed in the header is set. The ID property is used to display PlannerItems on the respective Positions and must match the PlannerItem's ResourceID property. The Size property is used to set the width of the Position.

Example:

```
Planner1.Positions.Clear();
PlannerPosition pp = new PlannerPosition(Planner1);

pp.Name = "Position 1";
pp.ID = "0";
pp.Size = 200;
Planner1.Positions.Add(pp);

pp = new PlannerPosition(Planner1);
pp.Name = "Position 2";
pp.ID = "1";
pp.Size = 100;
Planner1.Positions.Add(pp);
```

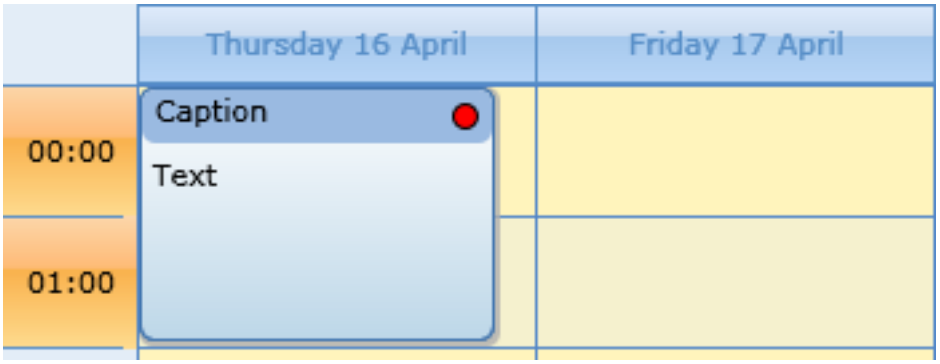


C) **MultiDay**

To let the Planner display multiple days, set the DayMapping and the NumberOfDays properties. The ColWidth property can be used to set the width of all Columns at once.

Example:

```
Planner1.DayMapping = Planner.DayMapType.MultiDay;
Planner1.NumberOfDays = 2;
Planner1.ColWidth = 150;
```



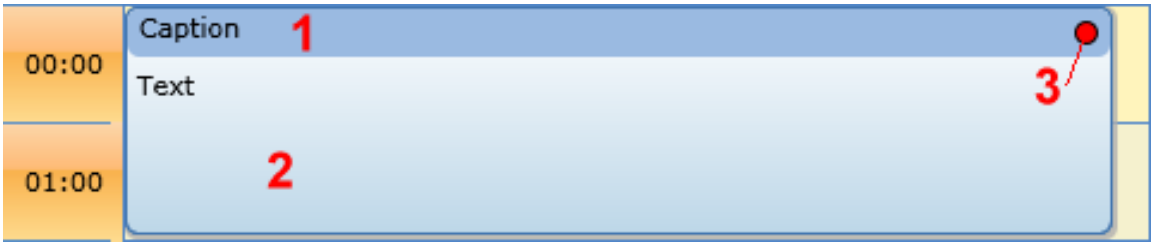
Adding Items

A) **Programmatically**

Below you can see how to add items in code and in some cases extra properties can be set to change the item's appearance.

```
Planner1.BeginUpdate();
PlannerItem pi = new PlannerItem();
pi.StartTime = DateTime.Today;
pi.EndTime = pi.StartTime.AddHours(2);
pi.CaptionText = "Caption";
pi.Text = "Text";
Planner1.Items.Add(pi);
```

```
Planner1.EndUpdate();
```



1) Caption
2) Notes
3) Delete button

When using a MultiResource Planner, an additional property ResourceID must be set to the value of the respective Position.ID property on which the PlannerItem should be displayed.

B) **Via Databinding**

The Planner can be databound to any storage type, for example a SQL Server database to dynamically display PlannerItems. (See TMS Silverlight Planner databinding or TMS WPF Planner databinding)

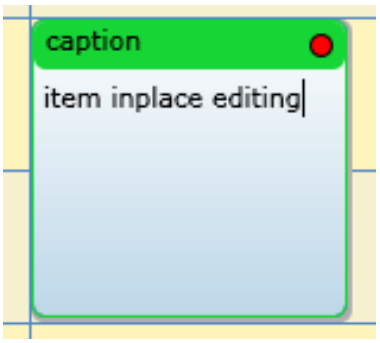
Note: In design-time the Silverlight - WPF Planner does not display any items.

Edit Modes

One click on an item, selects the item. A click on a selected item sets the item in edit mode. Ending the edit mode will set the item back to it's selected state, displaying any changes made to the item's content. There are three possible edit modes:

A) Inplace

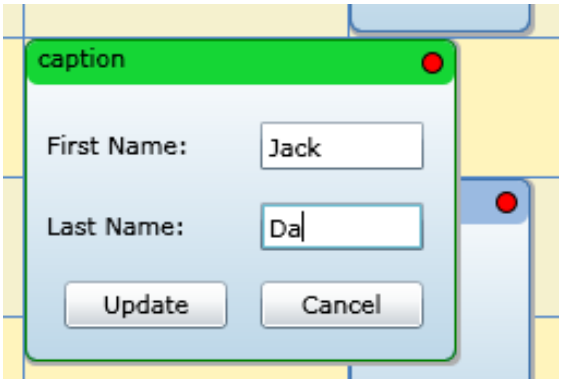
This is the default edit mode. The item's text can be edited using a TextBox displayed inplace in the PlannerItem's Notes area. Exit the editmode by pressing the TAB-key or clicking anywhere outside the item.



B) Detail

When editing starts, the event is resized with an animation to the size defined in the

DetailEditHeight and DetalEditWidth properties and the ItemEditingContent content is displayed.
The content is a completely customisable DataTemplate and can contain user-defined form controls bound to a database field.



Example:

The ItemEditingContent DataTemplate XAML is placed inside the Planner tags:

```
            <tms:Planner.ItemEditingContent>
                <DataTemplate>
                    <Grid Margin="5">
                        <Grid.RowDefinitions>
                            <RowDefinition></RowDefinition>
                            <RowDefinition></RowDefinition>
                            <RowDefinition></RowDefinition>
                        </Grid.RowDefinitions>
                        <Grid.ColumnDefinitions>
                            <ColumnDefinition></ColumnDefinition>
                            <ColumnDefinition></ColumnDefinition>
                        </Grid.ColumnDefinitions>
                        <TextBlock Text="First Name:" Grid.Row="0"
Grid.Column="0" VerticalAlignment="Center" />
                        <TextBlock Text="Last Name:"  Grid.Row="1"
Grid.Column="0" VerticalAlignment="Center" />
                        <TextBox Grid.Row="0" Grid.Column="1" Width="75"
Height="22" Text="{Binding FirstName, Mode=TwoWay}" />
                        <TextBox Grid.Row="1" Grid.Column="1" Width="75"
Height="22" Text="{Binding LastName, Mode=TwoWay}" />
                        <Button Grid.Row="2" Grid.Column="0"
x:Name="BtUpdate" Content="Update" Width="75" Height="22"
Click="BtUpdate_Click" />
                        <Button Grid.Row="2" Grid.Column="1"
x:Name="BtCancel" Content="Cancel" Width="75" Height="22"
Click="BtCancel_Click" />
                    </Grid>
                </DataTemplate>
            </tms:Planner.ItemEditingContent>
```

The code associated with the Update and Cancel button click events:

```
        private void BtUpdate_Click(object sender, RoutedEventArgs e)
        {
```

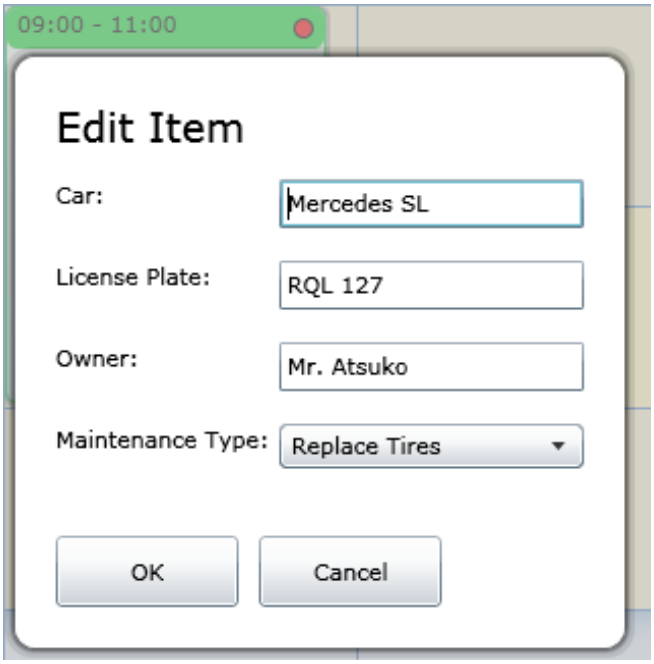
```
            PlannerItem pi = TestPlanner1.Items[TestPlanner1.SelectedIndex]
as PlannerItem;

            TestPlanner1.Databinding.UpdateDetailEdit(pi);
            TestPlanner1.EndEditing();
        }

        private void BtCancel_Click(object sender, RoutedEventArgs e)
        {
            TestPlanner1.EndEditing();
        }
```

C) PopupDetail

The Planner UI is temporarily disabled and a custom form is displayed above the item that is being edited.



Example:

When using the PopupDetail edit mode, an additional custom form is required. This example uses a form called PopupDetail (see picture above).

First create a new PopupDetail object (this is typically a custom form), optionally assign the OKClick and CancelClick events, and add it to the LayoutRoot object of the project:

```
PopupDetail editform = new PopupDetail();
editform.OKClick += new PopupDetail.ClickHandler(editform_OKClick);
editform.CancelClick += new PopupDetail.ClickHandler(editform_CancelClick);
LayoutRoot.Children.Add(editform);
```

Then assign the ItemEditing event to display the custom detail edit form:

```
        void TMSPlanner_ItemEditing(object sender,
Planner.PlannerItemOffsetEventArgs e)
        {
            if (TMSPlanner.EditMode == Planner.EditModes.PopupDetail)
                editform.EditItem(e.Item, e.Offset);
        }
```

The optional OKClick and CancelClick events can be configured as shown:

```
        void editform_CancelClick(object sender, PopupDetail.ItemEventArgs
e)
        {
            e.Item.GoToSelectedState(true);
            TMSPlanner.EditIndex = -1;
        }

        void editform_OKClick(object sender, PopupDetail.ItemEventArgs e)
        {
            TMSPlanner.EditIndex = -1;
            TMSPlanner.Databinding.Update(e.Item);
        }
```

Customizing the Header or HeaderGroup

To add custom elements to the Planner's Header or HeaderGroup, the GetHeaderCell or GetGroupCell methods can be used.

Example:

```
Grid hc = Planner1.GetHeaderCell(ResourceIndex, true);
hc.Children.Add(MyElement);
```

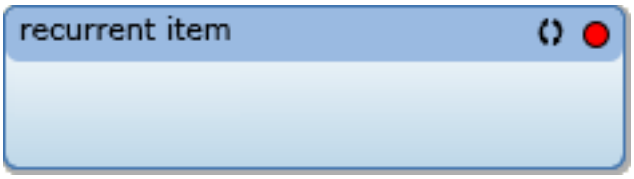
Adding text to the TimeAxis or a Grid cell

To add custom text to the Planner's TimeAxis or one of the cells in the Grid, use the Planner's GetTimeAxisText or GetCellText events.

Example:

```
        private void Planner1_GetCellText(object sender,
Planner.CellTextEventArgs e)
        {
            if (e.Row == 4)
                e.Text = "Lunch Break";
        }
```

Recurrent Items

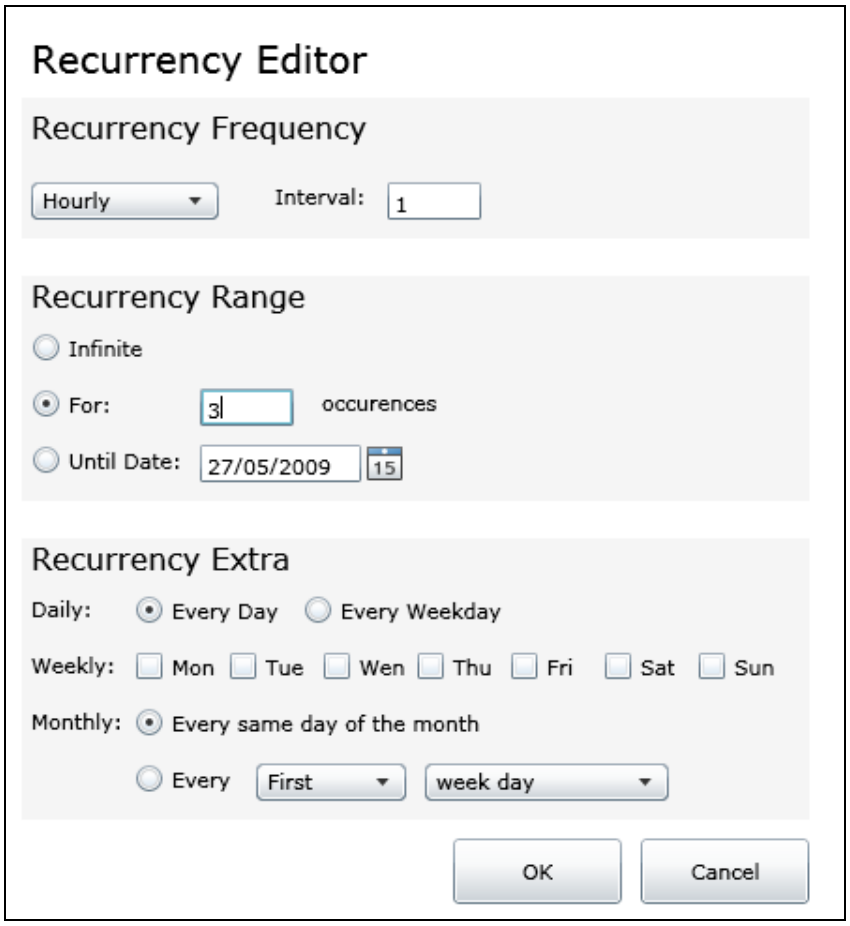


The Silverlight - WPF Planner has built-in support for recurrency. Recurrent events can be mixed with non recurring events. If an event is recurring, this means that an event in a single database record can have multiple occurrences on the Planner. Reucurrency is specified via a recurrency formula that is based on an implementation of the RFC 2445 iCalendar spec.
The recurrency formula is a string and therefore stored as a stringfield in the database. Note that recurrency formulas can become long for complex recurrency specifications. It is therefore recommended (but not mandatory) to use a memo field. For simple recurrencies without recurrency exceptions, a 60 character field should be sufficient. Note that for scheduling applications that do not need recurrency, no recurrency field is required in the database.
In addition to the recurrency field, 2 additional fields are required: FirstStartTime and FirstEndTime. These fields hold the start time of the first occurrence of the event and the end time of the first occurrence of the event.

As a visual indicator for a recurrent item, by default a recurrency indicator icon is displayed in the top right corner of the item caption next to the delete button. The PlannerItem's RecurrencyGlyph property can be set with a path to an image file if a custom recurrency indicator icon is required. A recurrency editor can be displayed when the recurrency indicator is clicked, by assigning the ItemRecurrencyClick event.

Recurrency Editor

The Planner comes with a recurrency editor so it's not necessary to write recurrency formulas directly but you can build your own editor as well.



Example:

First create a new InternalRecurrency object (this is the recurrency editor form included with the Planner) and add it to the LayoutRoot object of the project:

```
InternalRecurrency recform = new InternalRecurrency();
LayoutRoot.Children.Add(recform);
```

Then assign the ItemRecurrencyClick event to display the recurrency editor:

```
private void Planner_ItemRecurrencyClick(object sender,
Planner.PlannerItemEventArgs e)
{
    recform.Display(e.Item);
}
```

The Item's recurrency rule is automatically updated when the OK button is clicked. The form implements an OKClick event and a CancelClick event that fire when the respective buttons are clicked.

Recurrency Rules

To programmatically create recurrent items, the structure of the recurrency string is built from a series of specifiers SPECIFIER=value, separated by a ';' delimiter. The rule is prefixed by RRULE: while the exceptions are prefixed by EXDATES:

Example:
```
RRULE:FREQ=MONTHLY;COUNT=5;BYDAY=2WE
```

Possible specifiers:

Frequency

FREQ=MONTHLY

The frequency specifier can be: HOURLY, DAILY, WEEKLY, MONTHLY, YEARLY

Interval

INTERVAL=2

The default interval is 1. When the interval is 1, this does not need to be specified in the recurrency formula. The interval sets the number of time blocks between two recurring events. If the interval is set to 2 for a hourly recurring event of 1 hour duration, this will create a recurring series of events with one hour blocks between the events.

Count

COUNT=3

Sets the number of occurrences of the event

Until

UNTIL=end date

Sets the date of the last occurrence of the item in the recurrent series. Note that the time

specification is in ISO format, ie: YYYYMMDD'T'HHMMSS. So, Sep 21, 2009 16h30 is written as 20090921T163000

Note: a recurrency without COUNT or UNTIL specifier is considered an infinite recurrency.

ByDay, ByMonth

BYDAY=series of days

BYMONTH=series of months

The ByDay and ByMonth specifiers set for which days or months the recurrency rule is applicable. The series of days or months is a comma delimited series of the names of days using the 2 first letters for days or month numbers.

BYDAY=MO,TU,WE,TH,FR,SA,SU
BYMONTH=1,2,3,4,5,6,7,8,9,10,11,12

Example:
RRULE:FREQ=WEEKLY;COUNT=9;BYDAY=TU,TH

This rule specifies a recurrence for 9 occurrences weekly repeated on every Tuesday and Thursday of the week.

RRULE:FREQ=DAILY;UNTIL=20091231T000000;BYMONTH=1,2

This rule specifies daily recurring events during the months January and February till Dec 31, 2005.

Optional specifier for ByDay

An optional specifier can be used for the day of weeks to indicate in what day of a month an event should occur. This is done by prefixing the day name by the occurrency number of the day in the month.
Example:
RRULE:FREQ=MONTHLY;COUNT=5;BYDAY=2WE

This specifies a recurrent event, every month, for 5 months on the 2nd Wednesday of the month.

Exceptions

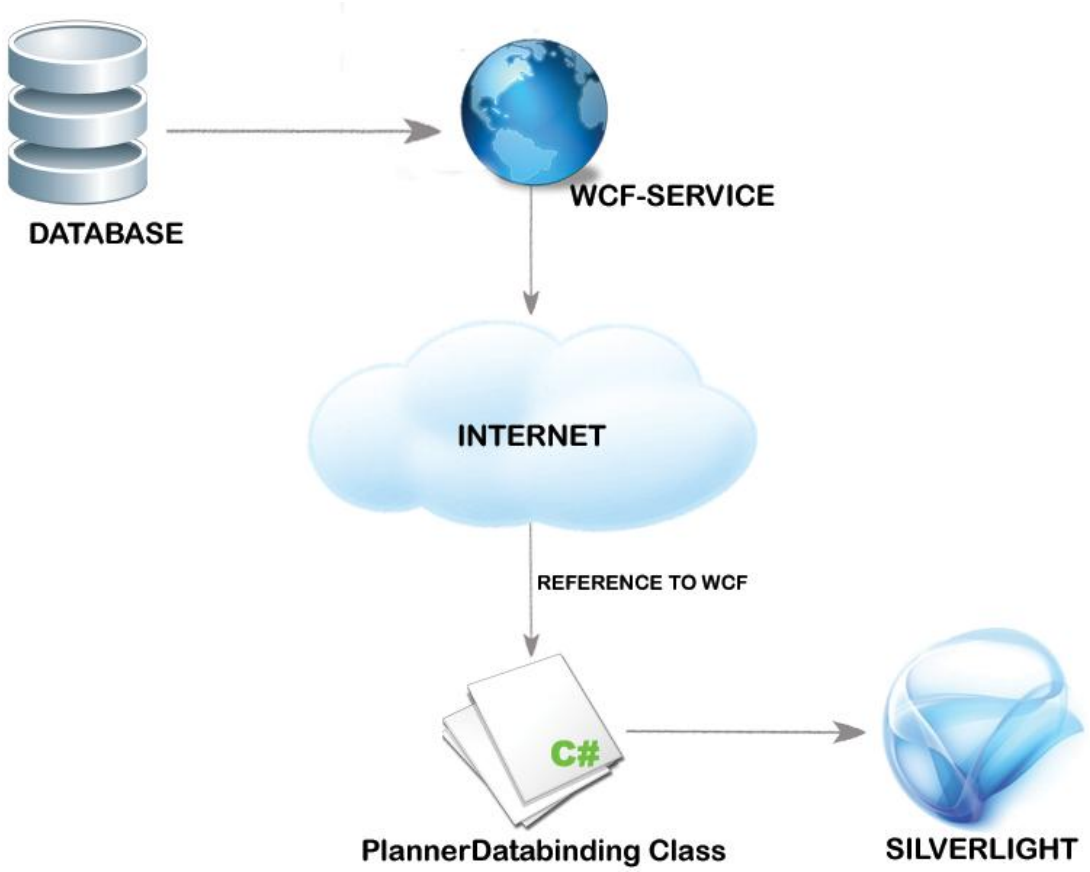
Exceptions to recurrency rules are a list of dates for which the rule is not applicable. The exceptions can be specified by adding these to the recurrency string as comma delimited ISO start and end dates:

EXDATES:startdate1/enddate1,startdate2/enddate2, ….

Example:

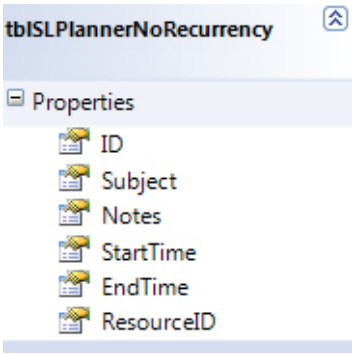
EXDATES:20040913T000000/20090913T235959,20090914T000000/20090914T235959

TMS Silverlight - WPF Planner databinding



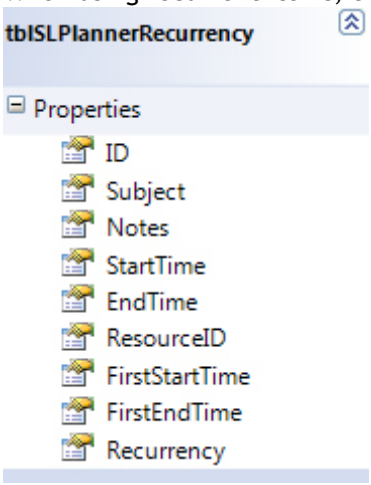
How to databind a Silverlight Planner

A database is required with the following fields:



ID (primary key, int)
Subject (varchar)
Notes (varchar)
StartTime (datetime)
EndTime (datetime)
ResourceID (int)

When using recurrent items, three more fields are required:



FirstStartTime (datetime)
FirstEndTime (datetime)
Recurrency (varchar)

The WCF service

A Silverlight enabled WCF service needs to be added to your web-project and 4 operations need to be implemented: GetItems (Read), InsertItem, UpdateItem and DeleteItem.

The GetItems method returns a list of all requested records.
The InsertItem method returns the ID of the inserted record. The return type depends on the type of your ID field in the database. For example: this can be an integer if it's an autoincrement field or a string if it's a GUID field.

A service reference to the WCF service needs to be added to your Silverlight project.

Example:

```
        DataContext datacontext = new
DataContext("Password=password;Persist Security Info=True;User
ID=userid;Initial Catalog=databasename;Data Source=slqserver");

       [OperationContract]
        public List<SLPlannerItem> GetItems(DateTime StartTime, DateTime
EndTime, String User)
        {
            return (from Items in datacontext.SLPlannerItemDemo1s
                    where
                    (
                    (Items.UserName.Equals(User)) &
                    (
                    (Items.StartTime <= StartTime && Items.EndTime >=
StartTime)
                    ||
                    (Items.StartTime <= EndTime && Items.EndTime >=
EndTime)
                    ||
                    (Items.StartTime >= StartTime && Items.EndTime <=
EndTime)
```

```
                    )
                    )
                    select Items).ToList();
        }

        [OperationContract]
        public int InsertItem(SLPlannerItem newRecord)
        {
            datacontext.SLPlannerItemDemo1s.InsertOnSubmit(newRecord);
            datacontext.SubmitChanges();
            return newRecord.ID;
        }

        [OperationContract]
        public void DeleteItem(SLPlannerItem deletedRecord)
        {
            PlannerItem recordToDelete =
datacontext.SLPlannerItemDemo1s.Single(r => r.ID == deletedRecord.ID);

            datacontext.SLPlannerItemDemo1s.DeleteOnSubmit(recordToDelete);
            datacontext.SubmitChanges();
        }

        [OperationContract]
        public void UpdateItem(SLPlannerItem updatedRecord)
        {
            SLPlannerItem oldRecord =
datacontext.SLPlannerItemDemo1s.Single(r => r.ID == updatedRecord.ID);

            if (oldRecord != null)
            {
                oldRecord.Subject = updatedRecord.Subject;
                oldRecord.Notes = updatedRecord.Notes;
                oldRecord.StartTime = updatedRecord.StartTime;
                oldRecord.EndTime = updatedRecord.EndTime;

                if (updatedRecord.FirstStartTime != DateTime.MinValue)
                    oldRecord.FirstStartTime =
updatedRecord.FirstStartTime;

                if (updatedRecord.FirstEndTime != DateTime.MinValue)
                    oldRecord.FirstEndTime = updatedRecord.FirstEndTime;

                oldRecord.ResourceID = updatedRecord.ResourceID;
                oldRecord.Recurrency = updatedRecord.Recurrency;

                datacontext.SubmitChanges();
            }
        }
```

Multi-user WCF Service

In a multi-user environment, where different users are working with the same set of PlannerItems at the same time, it can be necessary to immediately reflect changes made by one user in the view of another user.
Therefore the existing WCF service methods must be extended and an additional IsDirty method must be added.

Example:

```
        DataContext datacontext = new
DataContext("Password=password;Persist Security Info=True;User
ID=userid;Initial Catalog=databasename;Data Source=slqserver");

        public struct ReturnList
        {
            private List<SLPlannerItem> _list;
            private DateTime _mostRecentChange;

            public ReturnList(List<SLPlannerItem> list, DateTime
mostRecentChange)
            {
                _list = list;
                _mostRecentChange = mostRecentChange;
            }

            public List<SLPlannerItem> List
            {
                get { return _list; }
                set { _list = value; }
            }

            public DateTime MostRecentChange
            {
                get { return _mostRecentChange; }
                set { _mostRecentChange = value; }
            }
        }

        public struct ReturnValues
        {
            private bool _isDirty;
            private DateTime _mostRecentChange;

            public ReturnValues(bool isDirty, DateTime mostRecentChange)
            {
                _isDirty = isDirty;
                _mostRecentChange = mostRecentChange;
            }

            public bool IsDirty
            {
                get { return _isDirty; }
                set { _isDirty = value; }
            }

            public DateTime MostRecentChange
            {
                get { return _mostRecentChange; }
                set { _mostRecentChange = value; }
            }
        }

        public struct NewReturnValues
        {
```

```
            private bool _isDirty;
            private DateTime _mostRecentChange;
            private int _ID;

            public NewReturnValues(bool isDirty, DateTime mostRecentChange,
int ID)
            {
                _isDirty = isDirty;
                _mostRecentChange = mostRecentChange;
                _ID = ID;
            }

            public bool IsDirty
            {
                get { return _isDirty; }
                set { _isDirty = value; }
            }

            public DateTime MostRecentChange
            {
                get { return _mostRecentChange; }
                set { _mostRecentChange = value; }
            }

            public int ID
            {
                get { return _ID; }
                set { _ID = value; }
            }
        }

        private List<SLPlannerItem> FillList(DateTime startTime, DateTime
endTime)
        {
            return (from Items in datacontext.SLPlannerItems
                    where
                    (
                    ((Items.Deleted.Value == false) || (Items.Deleted.Value
== null))
                    &&
                    (
                    (Items.StartTime <= startTime && Items.EndTime >=
startTime)
                    ||
                    (Items.StartTime <= endTime && Items.EndTime >=
endTime)
                    ||
                    (Items.StartTime >= startTime && Items.EndTime <=
endTime)
                    )
                    )
                    select Items).ToList();
        }

        private List<SLPlannerItem> FillListComplete(DateTime startTime,
DateTime endTime)
        {
            return (from Items in datacontext.SLPlannerItems
```

```
                    where
                    (
                    (Items.StartTime <= startTime && Items.EndTime >=
startTime)
                    ||
                    (Items.StartTime <= endTime && Items.EndTime >=
endTime)
                    ||
                    (Items.StartTime >= startTime && Items.EndTime <=
endTime)
                    )
                    select Items).ToList();
        }

        private bool CheckIfDirty(DateTime mostRecentChange, DateTime
startTime, DateTime endTime)
        {
            bool result = false;
            List<SLPlannerItem> check = FillListComplete(startTime,
endTime);

            foreach (SLPlannerItem pi in check)
            {
                if (pi.Modified != null)
                {
                    if (DateTime.Compare(pi.Modified.Value,
mostRecentChange) > 0)
                        result = true;
                }
            }
            return result;
        }

        [OperationContract]
        public ReturnList GetItems(DateTime startTime, DateTime endTime)
        {
            return new ReturnList(FillList(startTime, endTime),
DateTime.Now);
        }

        [OperationContract]
        public NewReturnValues InsertItem(SLPlannerItem newRecord, DateTime
mostRecentChange, DateTime startTime, DateTime endTime)
        {
            newRecord.Modified = DateTime.Now;
            datacontext.SLPlannerItems.InsertOnSubmit(newRecord);
            datacontext.SubmitChanges();

            return new NewReturnValues(CheckIfDirty(mostRecentChange,
startTime, endTime), newRecord.Modified.Value, newRecord.ID);
        }

        [OperationContract]
        public ReturnValues DeleteEvent(SLPlannerItem deletedRecord,
DateTime mostRecentChange, DateTime startTime, DateTime endTime)
        {
            bool isDirty = CheckIfDirty(mostRecentChange, startTime,
endTime);
```

```
            //Round the datetime to 10 Milliseconds
            //Accuracy of ASP.NET is greater than SQL server
            DateTime newRecentChange = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, DateTime.Now.Day,
                DateTime.Now.Hour, DateTime.Now.Minute,
DateTime.Now.Second, Int32.Parse(DateTime.Now.ToString("ff") + "0"));

            SLPlannerItem oldRecord = datacontext.SLPlannerItems.Single(r
=> r.ID == deletedRecord.ID);

            if (oldRecord != null)
            {
                oldRecord.Deleted = true;
                oldRecord.Modified = newRecentChange;
            }

            datacontext.SubmitChanges();

            return new ReturnValues(isDirty, newRecentChange);
        }

        [OperationContract]
        public ReturnValues UpdateEvent(SLPlannerItem updatedRecord,
DateTime mostRecentChange, DateTime startTime, DateTime endTime)
        {
            bool isDirty = CheckIfDirty(mostRecentChange, startTime,
endTime);

            //Round the datetime to 10 Milliseconds
            //Accuracy of ASP.NET is greater than SQL server
            DateTime newRecentChange = new DateTime(DateTime.Now.Year,
DateTime.Now.Month, DateTime.Now.Day,
                DateTime.Now.Hour, DateTime.Now.Minute,
DateTime.Now.Second, Int32.Parse(DateTime.Now.ToString("ff") + "0"));

            SLPlannerItem oldRecord = datacontext.SLPlannerItems.Single(r
=> r.ID == updatedRecord.ID);

            if (oldRecord != null)
            {
                oldRecord.Subject = updatedRecord.Subject;
                oldRecord.Notes = updatedRecord.Notes;
                oldRecord.StartTime = updatedRecord.StartTime;
                oldRecord.EndTime = updatedRecord.EndTime;

                if (updatedRecord.FirstStartTime != DateTime.MinValue)
                    oldRecord.FirstStartTime =
updatedRecord.FirstStartTime;

                if (updatedRecord.FirstEndTime != DateTime.MinValue)
                    oldRecord.FirstEndTime = updatedRecord.FirstEndTime;

                oldRecord.ResourceID = updatedRecord.ResourceID;
                oldRecord.Recurrency = updatedRecord.Recurrency;

                oldRecord.Modified = newRecentChange;
            }
```

```
            datacontext.SubmitChanges();

            return new ReturnValues(isDirty, newRecentChange);
        }

        [OperationContract]
        public ReturnValues IsDirty(DateTime mostRecentChange, DateTime
startTime, DateTime endTime)
        {
            bool isDirty = CheckIfDirty(mostRecentChange, startTime,
endTime);

            if (isDirty)
                mostRecentChange = DateTime.Now;

            return new ReturnValues(isDirty, mostRecentChange);
        }
```

The Silverlight PlannerDatabinding Class

The Planner implements a base PlannerDatabinding class. A new class, derived from PlannerDatabinding, needs to be added to the project and the four methods need to be overridden. This makes it possible to connect to any type of serverside hosted database.
The four methods in the PlannerDatabinding class allow for all interaction the Planner needs to perform with the Database: Insert, Update, Delete and Read.

The list of requested records is returned when the GetItemsCompleted event is called.
The InsertItemCompleted event returns the ID of the record that has just been added to the database.

Example:

```
    public class WCFPlannerDatabinding : TMSSLControls.PlannerDatabinding
    {
        #region Variables
        ObservableCollection<SLPlannerItem> itemsSource;
        PlannerServiceClient svc = new PlannerServiceClient();
        Planner dbPlanner;
        PlannerItem insertedItem;
        #endregion Variables

        #region Constructor
        public WCFPlannerDatabinding(Planner planner)
        {
            dbPlanner = planner;
            svc.InsertItemCompleted += new
EventHandler<InsertItemCompletedEventArgs>(svc_InsertItemCompleted);
            svc.GetItemsCompleted += new
EventHandler<GetItemsCompletedEventArgs>(svc_GetItemsCompleted);
        }

        #endregion Constructor

        #region Methods
```

```
        #region GetItemsSource
        public ObservableCollection<SLPlannerItem> GetItemsSource()
        {
              //Comment the line below when using Microsoft Visual Studio
2008 with Silverlight 2 or 3
              return new
ObservableCollection<SLPlannerItem>((System.Collections.Generic.IEnumerable
<SLPlannerItem>)dbPlanner.ItemsSource);

              //Comment the line below when using Microsoft Visual Studio
2010 with Silverlight 4
              return
(ObservableCollection<SLPlannerItem>)dbPlanner.ItemsSource;

        }
        #endregion

        #region Insert
        public override void Insert(PlannerItem plannerItem)
        {
            SLPlannerItem newItem = new SLPlannerItem();
            newItem.StartTime = plannerItem.StartTime;
            newItem.ResourceID = Int32.Parse(plannerItem.ResourceID);
            newItem.EndTime = plannerItem.EndTime;
            newItem.Notes = plannerItem.Text;
            newItem.Subject = plannerItem.CaptionText;

            //immediately assign the new planneritem's datacontext
            plannerItem.DataContext = newItem;
            insertedItem = plannerItem;

            svc.InsertItemAsync(newItem);
        }
        #endregion Insert

        #region Update
        public override void Update(PlannerItem plannerItem)
        {
            SLPlannerItem selectedItem = plannerItem.DataContext as
SLPlannerItem;

            if (plannerItem.Recurrency.Length > 0)
            {
                selectedItem.StartTime = plannerItem.RecurrencyStartTime;
                selectedItem.EndTime = plannerItem.RecurrencyEndTime;
                selectedItem.FirstStartTime =
plannerItem.RecurrencyStartTimeFirst;
                selectedItem.FirstEndTime =
plannerItem.RecurrencyEndTimeFirst;
            }
            else
            {
                selectedItem.StartTime = plannerItem.StartTime;
                selectedItem.EndTime = plannerItem.EndTime;
            }

            selectedItem.ResourceID = Int32.Parse(plannerItem.ResourceID);
            selectedItem.Notes = plannerItem.Text;
```

```
            selectedItem.Subject = plannerItem.CaptionText;
            selectedItem.Recurrency = plannerItem.Recurrency;

            svc.UpdateItemAsync(selectedItem);

            //Update the planner's itemssource
            itemsSource = GetItemsSource();
            ItemsSource[itemsSource.IndexOf(selectedItem)] = selectedEvent;
            dbPlanner.ItemsSource = null;
            dbPlanner.ItemsSource = itemsSource;

        }
        #endregion Update

        #region Delete
        public override void Delete(PlannerItem plannerItem)
        {
            svc.DeleteItemAsync(plannerItem.DataContext as SLPlannerItem);

            SLPlannerItem selectedItem = plannerItem.DataContext as
SLPlannerItem;

            //Update the planner's itemssource
            itemsSource = GetItemsSource();
            itemsSource.Remove(selectedItem);
            dbPlanner.ItemsSource = null;
            dbPlanner.ItemsSource = itemsSource;
        }
        #endregion Delete

        #region Read
        public override void Read()
        {
            svc.GetItemsAsync(dbPlanner.StartTime, dbPlanner.EndTime);
        }
        #endregion Read

        #endregion Methods

        #region Events


        void svc_GetItemsCompleted(object sender,
GetItemsCompletedEventArgs e)
        {
            if (e.Error == null)
                dbPlanner.ItemsSource = e.Result;
        }

        void svc_InsertItemCompleted(object sender,
InsertItemCompletedEventArgs e)
        {
            if ((e.Error == null) && (insertedItem != null))
            {
                SLPlannerItem selectedItem = insertedItem.DataContext as
SLPlannerItem;
                selectedItem.ID = e.Result;
```

```
                //Update the planner's itemssource
                itemsSource = GetItemsSource();
                itemsSource.Add(selectedEvent);
                dbPlanner.ItemsSource = null;
                dbPlanner.ItemsSource = itemsSource;

                insertedItem = null;
            }

        }

        #endregion Events
    }
```

DetailEdit databinding

When using the DetailEdit edit mode an additional method is available called UpdateDetailEdit. This method must be executed when changes need to be saved to the database. (For example when the DetailEdit form's Update button is clicked)

```
        #region UpdateDetailEdit
        public override void UpdateDetailEdit(PlannerItem plannerItem)
        {
            SLPlannerItem selectedEvent = plannerItem.DataContext as
SLPlanerItem;
            svc.UpdateEventAsync(selectedEvent);
        }
        #endregion UpdateDetailEdit
```

Multi-user databinding

In a multi-user environment, where different users are working with the same set of PlannerItems at the same time, it can be necessary to immediately reflect changes made by one user in the view of another user.
When setting the MultiUser property to True, the Planner will call the DataBinding.IsDirty method every 5 seconds.
The IsDirtyCompleted event will then return the timestamp of the most recent change of an item and a boolean IsDirty indicating if the PlannerItems need to be refreshed.
Optionally the CheckIfDirty method can also be called when the UpdateEventCompleted and/or the DeleteEventCompleted are fired.

Example:

```
    public class WCFPlannerDatabinding : TMSSLControls.PlannerDatabinding
    {
        #region Variables
        DataServiceClient svc = new DataServiceClient();
        Planner dbPlanner;
        PlannerItem insertedItem;
        DateTime mostRecentChange = DateTime.MinValue;
        #endregion Variables

        #region Constructor
        public WCFPlannerDatabinding(Planner planner)
```

```
        {
            dbPlanner = planner;
            svc.InsertItemCompleted += new
EventHandler<InsertItemCompletedEventArgs>(svc_InsertItemCompleted);
            svc.GetItemsCompleted += new
EventHandler<GetItemsCompletedEventArgs>(svc_GetItemsCompleted);
            svc.UpdateEventCompleted += new
EventHandler<UpdateEventCompletedEventArgs>(svc_UpdateEventCompleted);
            svc.DeleteEventCompleted += new
EventHandler<DeleteEventCompletedEventArgs>(svc_DeleteEventCompleted);
            svc.IsDirtyCompleted += new
EventHandler<IsDirtyCompletedEventArgs>(svc_IsDirtyCompleted);
        }
        #endregion Constructor

        #region Methods

        #region Insert
        public override void Insert(PlannerItem plannerItem)
        {
            SLPlannerItem newItem = new SLPlannerItem();
            newItem.StartTime = plannerItem.StartTime;
            newItem.ResourceID = Int32.Parse(plannerItem.ResourceID);
            newItem.EndTime = plannerItem.EndTime;
            newItem.Notes = plannerItem.Text;
            newItem.Subject = plannerItem.CaptionText;

            //immediately assign the new planneritem's datacontext
            plannerItem.DataContext = newItem;
            insertedItem = plannerItem;

            svc.InsertItemAsync(newItem, mostRecentChange,
dbPlanner.StartTime, dbPlanner.EndTime);

            dbPlanner.DisableUI();
        }
        #endregion Insert

        #region Update
        public override void Update(PlannerItem plannerItem)
        {
            SLPlannerItem selectedEvent = plannerItem.DataContext as
SLPlannerItem;

            if (plannerItem.Recurrency.Length > 0)
            {
                selectedEvent.StartTime = plannerItem.RecurrencyStartTime;
                selectedEvent.EndTime = plannerItem.RecurrencyEndTime;
                selectedEvent.FirstStartTime =
plannerItem.RecurrencyStartTimeFirst;
                selectedEvent.FirstEndTime =
plannerItem.RecurrencyEndTimeFirst;
            }
            else
            {
                selectedEvent.StartTime = plannerItem.StartTime;
                selectedEvent.EndTime = plannerItem.EndTime;
            }
```

```
            selectedEvent.ResourceID = Int32.Parse(plannerItem.ResourceID);
            selectedEvent.Notes = plannerItem.Text;
            selectedEvent.Subject = plannerItem.CaptionText;
            selectedEvent.Recurrency = plannerItem.Recurrency;

            svc.UpdateEventAsync(selectedEvent, mostRecentChange,
dbPlanner.StartTime, dbPlanner.EndTime);

            //Update the planner's itemssource
            ObservableCollection<SLPlannerItem> itemsSource =
(ObservableCollection<SLPlannerItem>)dbPlanner.ItemsSource;
            itemsSource[itemsSource.IndexOf(selectedEvent)] =
selectedEvent;
            dbPlanner.ItemsSource = null;
            dbPlanner.ItemsSource = itemsSource;
        }
        #endregion Update

        #region Delete
        public override void Delete(PlannerItem plannerItem)
        {
            svc.DeleteEventAsync(plannerItem.DataContext as SLPlannerItem,
mostRecentChange, dbPlanner.StartTime, dbPlanner.EndTime);

            SLPlannerItem selectedEvent = plannerItem.DataContext as
SLPlannerItem;

            //Update the planner's itemssource
            ObservableCollection<SLPlannerItem> itemsSource =
(ObservableCollection<SLPlannerItem>)dbPlanner.ItemsSource;
            itemsSource.Remove(selectedEvent);
            dbPlanner.ItemsSource = null;
            dbPlanner.ItemsSource = itemsSource;
        }
        #endregion Delete

        #region Read
        public override void Read()
        {
            svc.GetItemsAsync(dbPlanner.StartTime, dbPlanner.EndTime);
        }
        #endregion Read

        #region IsDirty
        public override void IsDirty()
        {
            svc.IsDirtyAsync(mostRecentChange, dbPlanner.StartTime,
dbPlanner.EndTime);
        }
        #endregion IsDirty

        #region CheckIfDirty
        private void CheckIfDirty(Demo4DataServiceReturnValues
returnvalues)
        {
            mostRecentChange = returnvalues.MostRecentChange;
```

```
            if (returnvalues.IsDirty)
                Read();
        }
        #endregion CheckIfDirty

        #endregion Methods

        #region Events

        #region InsertItemCompleted
        void svc_InsertItemCompleted(object sender,
InsertItemCompletedEventArgs e)
        {
            if ((e.Error == null) && (insertedItem != null))
            {
                Demo4DataServiceNewReturnValues result = e.Result;
                mostRecentChange = result.MostRecentChange;

                SLPlannerItem selectedEvent = insertedItem.DataContext as
SLPlannerItem;
                insertedItem = null;
                selectedEvent.ID = result.ID;

                //Update the planner's itemssource
                ObservableCollection<SLPlannerItem> itemsSource =
(ObservableCollection<SLPlannerItem>)dbPlanner.ItemsSource;
                itemsSource.Add(selectedEvent);
                dbPlanner.ItemsSource = null;
                dbPlanner.ItemsSource = itemsSource;

                if (result.IsDirty)
                    Read();
            }

            dbPlanner.EnableUI();
        }
        #endregion InsertItemCompleted

        #region GetItemsCompleted
        void svc_GetItemsCompleted(object sender,
GetItemsCompletedEventArgs e)
        {
            if (e.Error == null)
            {
                dbPlanner.ItemsSource = e.Result.List;
                mostRecentChange = e.Result.MostRecentChange;
            }
        }
        #endregion GetItemsCompleted

        void svc_UpdateEventCompleted(object sender,
UpdateEventCompletedEventArgs e)
        {
            if (e.Error == null)
                CheckIfDirty(e.Result);
        }
```

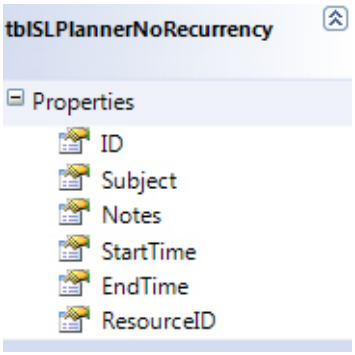
```
        void svc_DeleteEventCompleted(object sender,
DeleteEventCompletedEventArgs e)
        {
            if (e.Error == null)
                CheckIfDirty(e.Result);
        }

        void svc_IsDirtyCompleted(object sender, IsDirtyCompletedEventArgs
e)
        {
            if (e.Error == null)
                CheckIfDirty(e.Result);
        }

        #endregion Events
    }
```

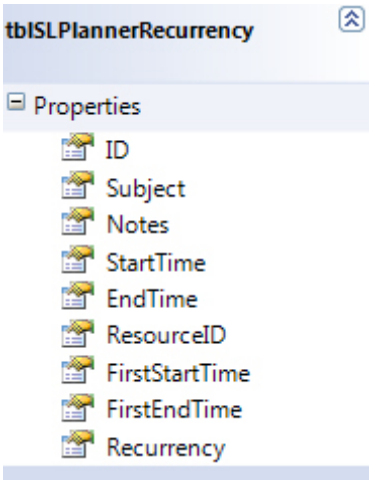
How to databind a WPF Planner

A database is required with the following fields:



ID (primary key, int)
Subject (varchar)
Notes (varchar)
StartTime (datetime)
EndTime (datetime)
ResourceID (int)

When using recurrent items, three more fields are required:



FirstStartTime (datetime)
FirstEndTime (datetime)
Recurrency (varchar)

The WPF PlannerDatabinding Class

The Planner implements a base PlannerDatabinding class. A new class, derived from PlannerDatabinding, needs to be added to the project and the four methods need to be overridden. This makes it possible to connect to any type of database.
The four methods in the PlannerDatabinding class allow for all interaction the Planner needs to perform with the Database: Insert, Update, Delete and Read.

When the Read method is called the Planner's DataContext property is assigned with the acquired DataSet data.

Example:

```
    class SQLPlannerDatabinding : PlannerDatabinding
    {
        #region Variables
        SqlConnection con = new SqlConnection();
        SqlDataAdapter ad = new SqlDataAdapter();
        DataSet ds = new DataSet();

        TMSSLControls.IPlanner dbPlanner;

        #endregion Variables

        #region Constructor
        public SQLPlannerDatabinding(TMSSLControls.IPlanner planner)
        {
            dbPlanner = planner;
            con.ConnectionString = "Data Source=SQLServerName; Initial
Catalog=DatabaseName; Integrated Security=True";


            String str = "SELECT StartTime, EndTime, FirstStartTime,
FirstEndTime, Subject, Notes, Recurrency, ResourceID, ID "
                + " FROM PlannerItems WHERE "
```

```
                + "( (StartTime <= '" +
dbPlanner.PerdiodStartDate.ToString("yyyy/MM/dd HH:mm") + "' AND EndTime >=
'" + dbPlanner.PeriodStartDate.ToString("yyyy/MM/dd HH:mm") + "') "
                + " OR (StartTime <= '" +
dbPlanner.PeriodEndDate.ToString("yyyy/MM/dd HH:mm") + "' AND EndTime >= '"
+ dbPlanner.PeriodEndDate.ToString("yyyy/MM/dd HH:mm") + "') "
                + " OR (Events.StartTime >= '" +
dbPlanner.PeriodStartDate.ToString("yyyy/MM/dd HH:mm") + "' AND
Events.EndTime <= '" + dbPlanner.PeriodEndDate.ToString("yyyy/MM/dd HH:mm")
+ "') )"
                ;

            ad.SelectCommand = new SqlCommand(str, con);
        }
        #endregion Constructor

        #region Methods

        #region Read
        public override void Read()
        {
            ds.Clear();
            ad.Fill(ds);
            con.Close();
            dbPlanner.DataContext = ds.Tables[0].DefaultView;
            dbPlanner.RefreshSQLPlannerDataBinding();
        }
        #endregion Read

        #region Insert
        public override void Insert(PlannerItem plannerItem)
        {
            SqlCommand cmd = new SqlCommand();

            if (plannerItem.CaptionText == null)
                plannerItem.CaptionText = string.Empty;
            if (plannerItem.Text == null)
                plannerItem.Text = string.Empty;

            cmd.CommandText = "INSERT INTO PlannerItems (StartTime,
EndTime, Subject, Notes, ResourceID) "
             + "VALUES (@StartTime, @EndTime, @Subject, @Notes,
@ResourceID)";
            cmd.Connection = con;

            cmd.Parameters.Add(new SqlParameter("StartTime",
plannerItem.StartTime));
            cmd.Parameters.Add(new SqlParameter("EndTime",
plannerItem.EndTime));
            cmd.Parameters.Add(new SqlParameter("Subject",
plannerItem.CaptionText));
            cmd.Parameters.Add(new SqlParameter("Notes",
plannerItem.Text));
            cmd.Parameters.Add(new SqlParameter("ResourceID",

            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
```

```
            Read();
        }
        #endregion Insert

        #region Update
        public override void Update(PlannerItem plannerItem)
        {
            SqlCommand cmd = new SqlCommand();

            if (plannerItem.CaptionText == null)
                plannerItem.CaptionText = string.Empty;
            if (plannerItem.Text == null)
                plannerItem.Text = string.Empty;

            cmd.CommandText = "UPDATE PlannerItems SET StartTime =
@StartTime, EndTime = @EndTime, Subject = @Subject, "
             + " Notes = @Notes, ResourceID = @ResourceID, Recurrency =
@Recurrency,"
             + " FirstStartTime = @FirstStartTime, FirstEndTime =
@FirstEndTime WHERE ID = @ID";
            cmd.Connection = con;

            if (plannerItem.Recurrency.Length > 0)
            {
                cmd.Parameters.Add(new SqlParameter("StartTime",
plannerItem.RecurrencyStartTime));
                cmd.Parameters.Add(new SqlParameter("EndTime",
plannerItem.RecurrencyEndTime));
                cmd.Parameters.Add(new SqlParameter("FirstStartTime",
plannerItem.RecurrencyStartTimeFirst));
                cmd.Parameters.Add(new SqlParameter("FirstEndTime",
plannerItem.RecurrencyEndTimeFirst));
            }
            else
            {
                cmd.Parameters.Add(new SqlParameter("StartTime",
plannerItem.StartTime));
                cmd.Parameters.Add(new SqlParameter("EndTime",
plannerItem.EndTime));
                cmd.Parameters.Add(new SqlParameter("FirstStartTime",
string.Empty));
                cmd.Parameters.Add(new SqlParameter("FirstEndTime",
string.Empty));
            }

            cmd.Parameters.Add(new SqlParameter("Subject",
plannerItem.CaptionText));
            cmd.Parameters.Add(new SqlParameter("Notes",
plannerItem.Text));
            cmd.Parameters.Add(new SqlParameter("ResourceID",
plannerItem.ResPos));
            cmd.Parameters.Add(new SqlParameter("ID", plannerItem.Key));
            cmd.Parameters.Add(new SqlParameter("Recurrency",
plannerItem.Recurrency));

            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
```

```
            Read();
        }
        #endregion Update

        #region Delete
        public override void Delete(PlannerItem plannerItem)
        {
            SqlCommand cmd = new SqlCommand();

            cmd.CommandText = "DELETE FROM PlannerItems WHERE ID = @ID";
            cmd.Connection = con;

            cmd.Parameters.Add(new SqlParameter("ID", plannerItem.Key));

            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            Read();
        }
        #endregion Delete

        #endregion Methods

    }
```

Setting the databinding properties

Through various properties the Planner needs to be instructed which dataset fields should be used to display the relevant PlannerItems and its properties and which PlannerDatabinding derived class should be used to interact with the dataset. The Field* properties need to be set to a string value matching the dataset field names.

Example:

```
            Planner1.BeginUpdate();

            //Assign the Field* properties
            Planner1.FieldKey = "ID";
            Planner1.FieldEndTime = "EndTime";
            Planner1.FieldStartTime = "StartTime";
            Planner1.FieldSubject = "Subject";
            Planner1.FieldNotes = "Notes";
            Planner1.FieldResource = "ResourceID";

            //Three extra Field* properties need to be set when using
recurrent items
            Planner1.FirstFirstStarTime = "FirstStartTime";
            Planner1.FieldFirstEndTime = "FirstEndTime";
            Planner1.FieldRecurrency = "Recurrency";

            //Assign the Databinding property with the derived
PlannerDatabinding class

            //When using the Silverlight Planner
            Planner1.Databinding = new WCFPlannerDatabinding(Planner1);
```

```
//When using the WPF Planner
Planner1.Databinding = new SQLPlannerDatabinding(Planner1);

Planner1.EndUpdate();
```

Note: When changing multiple Planner properties, it's recommended to call the planner's BeginUpdate and EndUpdate methods to optimize performance.

Manipulating databound PlannerItems

The Planner provides two events that can be used when PlannerItems retrieved from the database need to be dynamically modified in code. This can be used for example to set PlannerItem's from the past to readonly and apply a different color to make this visually clear. When the PlannerItem object was retrieved from the database, the Planner databinding will automatically sets all properties like StartTime, EndTime, Caption, Notes and after setting this, the ItemRetrieved event is triggered allowing to set other PlannerItem properties dynamically from code, like in the example of past items, the ReadOnly or Brush properties.

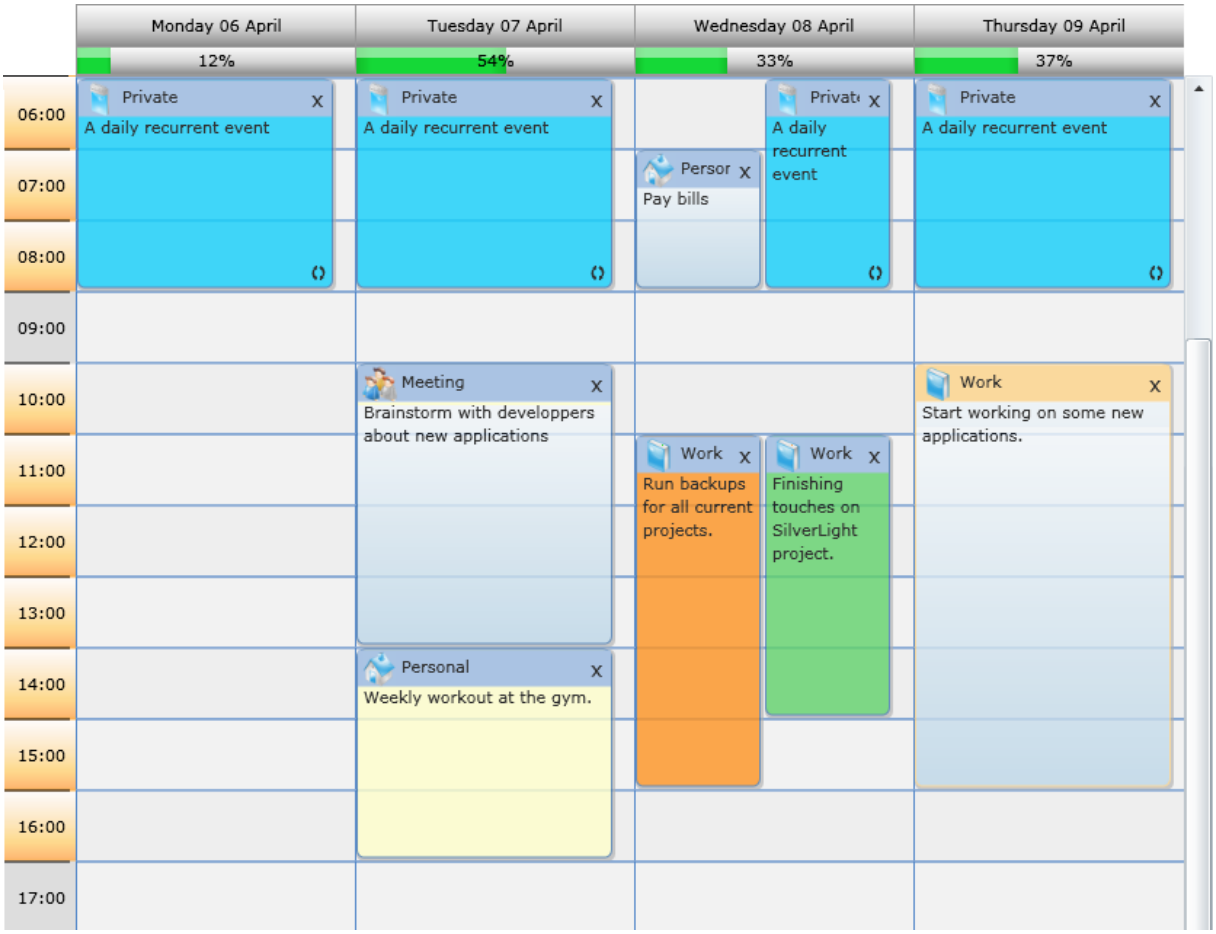
1) ItemRetrieved
   This event fires each time an Item has been retrieved from the dataset and added to the Planner's Items collection.

2) ItemsRetrieved
   This event fires after all items have been added to the Planner's Items collection.

Example:

```
private void Planner1_ItemsRetrieved(object sender, EventArgs e)
{
    foreach (PlannerItem item in Planner1.Items)
    {
        item.ReadOnly = true;
    }
}
```

Example of a TMS Silverlight - WPF Planner with custom colors, settings and databound PlannerItems (including recurrent items) with a custom style:



TMS Silverlight - WPF MonthPlanner use

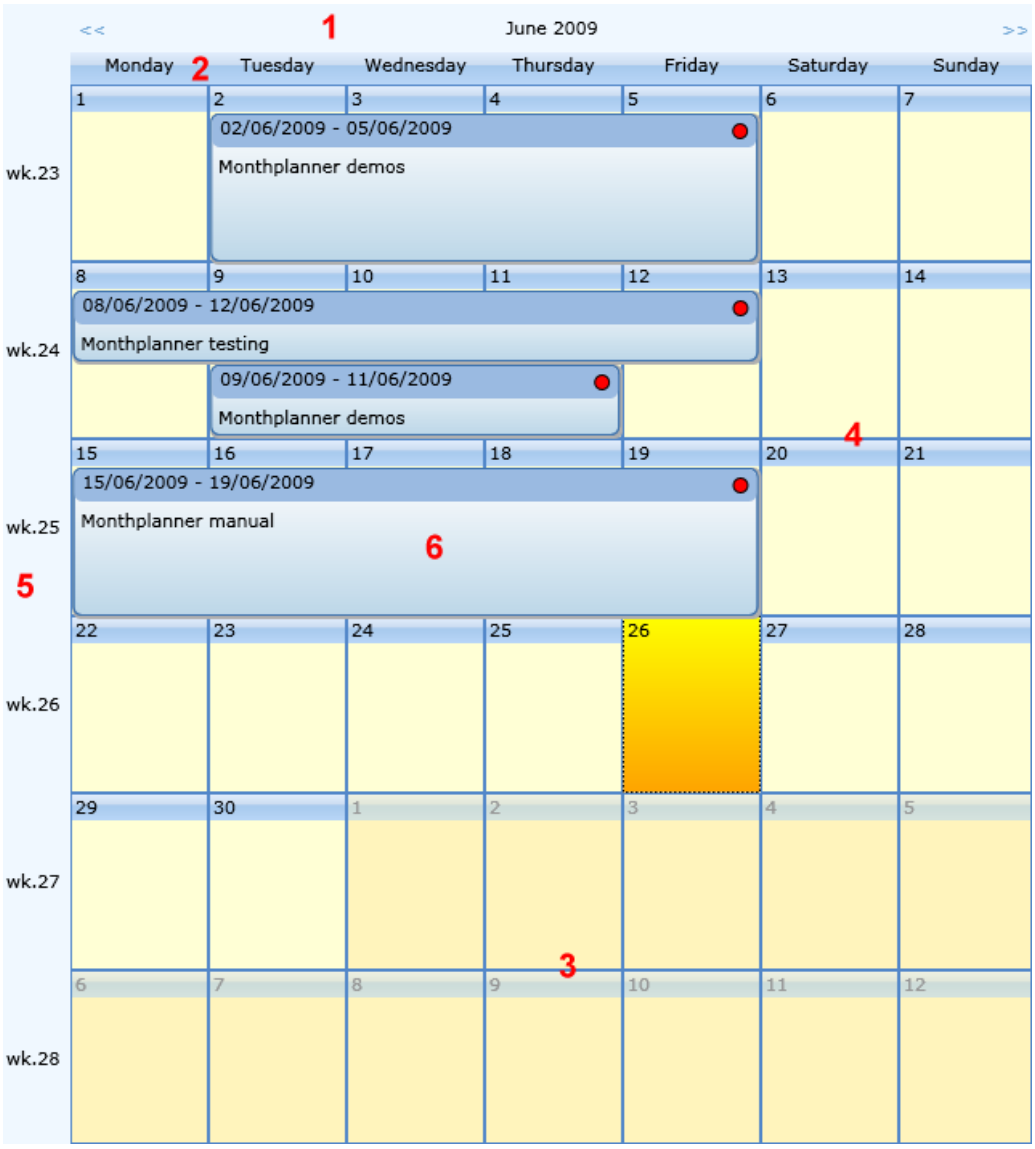
The MonthPlanner is similar in scope and functionality to the Planner control, except that it looks and feels like a wall calendar. Use the MonthPlanner to display events (e.g., meetings, appointments, classes) within any month of a given year. Events may span one or more days. The events may be programmatically added to the MonthPlanner or retrieved from a database via ItemsSource property. 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 the regular Planner is that the MonthPlanner has no resource concept. If an event is associated with a resource (e.g., class room, piece of equipment, person) then that relationship is retained however but the resource relationship does not influence how or where the event is rendered within the MonthPlanner control.

TMS Silverlight - WPF MonthPlanner organisation

The visual organisation of TMS Silverlight - WPF MonthPlanner

TMS Silverlight - WPF Planner is a component with following main visual elements:



1) **Header**: The Header area identifies the month and year for which events are being displayed. It also provides control for moving to the previous or next month.

2) **DayNames**: By default, there are 7 columns in the MonthPlanner, one for each day of the week. The DayNames identify the day associated witch each column.

3) **Other month days**: The MonthPlanner always displays six rows of days. The first and/or last row may display one or more days from the previous and/or following month. These areas will display events from those months and the user is allowed to drag existing events into those areas.

4) **Current month days**: The largest portion of the MonthPlanner is 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.

5) **WeekNumbers:** The column on the left side displays the index of the week in the current year for each row.

6) **PlannerItem**: The default PlannerItem has a caption, a notes section and a delete button. The PlannerItem class is identical to the class used in the Planner.

The programmatic organisation of the TMS Silverlight - WPF MonthPlanner

MonthPlanner-only properties overview

- ActiveMonthBackBrush
  - Brush, defines the background brush for days in the currently displayed month.
- ActiveMonthHeaderBackBrush
  - Brush, defines the header background brush for days in the currently displayed month.
- ActiveMonthTextBrush
  - Brush, defines the text brush for days in the currently displayed month.
- ColWeekNumbersBackBrush
  - Brush, defines the brush for the week numbers column.
- ColWeekNumbersVisible
  - Boolean, indicates if the column containing the week numbers is visible.
- ColWeekNumbersWidth
  - Integer, defines the width of the right-side column that contains the week index number.
- DayBorderBrush
  - Brush, defines the border brush for the MonthPlanner days.
- DayNamesBackBrush
  - Brush, defines the background brush for the DayNames row.
- DayNamesHeight
  - Integer, defines the height for the DayNames row.
- DayNamesTextBrush
  - Brush, defines the text brush for the DayNames row.
- FirstDayOfWeek
  - DayOfWeek, defines the first day of the week.
- HeaderBackBrush
  - Brush, defines the background brush for the Header.
- HeaderTextBrush
  - Brush, defines the text brush for the Header.
- HeaderVisible
  - Boolean, indicates if the header is visible.
- InactiveMonthBackBrush
  - Brush, defines the background brush for days in the previous or next month.
- InactiveMonthHeaderBackBrush

  - Brush, defines the header background brush for days in the previous or next month.

- InactiveMonthTextBrush
  - Brush, defines the text brush for days in the previous or next month.

- NextMonthButtonVisible
  - Boolean, indicates if the next month button is displayed in the title bar. Clicking this button will move the MonthPlanner's view to the previous month.

- PreviousMonthButtonVisible
  - Boolean, indicates if the previous month button is displayed in the title bar. Clicking this button will move the MonthPlanner's view to the next month.

- SelectionBrush
  - Brush, defines the background brush for selected days.

- ShowDaysBeforeMonth
  - Boolean, indicates if the days of the previous month may be displayed on the first row of the MonthPlanner.

- ShowDaysAfterMonth
  - Boolean, indicates if the days of the next month may be displayed on the last row of the MonthPlanner.

- ShowSelectionPreview
  - Boolean, indicates if the selection preview is displayed. If True, when dragging an PlannerItem the cells will be highlighted where the PlannerItem will be moved to when it's dropped.

- ShowToday
  - Boolean, indicates if today's date should be highlighted.

- StartMonth
  - DateTime, defines the which month of which year is displayed in the MonthPlanner.

- TodayFillBrush
  - Brush, defines the background brush for today's date when ShowToday is True.

- TodayStrokeBrush
  - Brush, defines the border brush for today's date when ShowToday is True.

- TodayStrokeThickness
  - Integer, defines the thickness of the stroke on today's date.

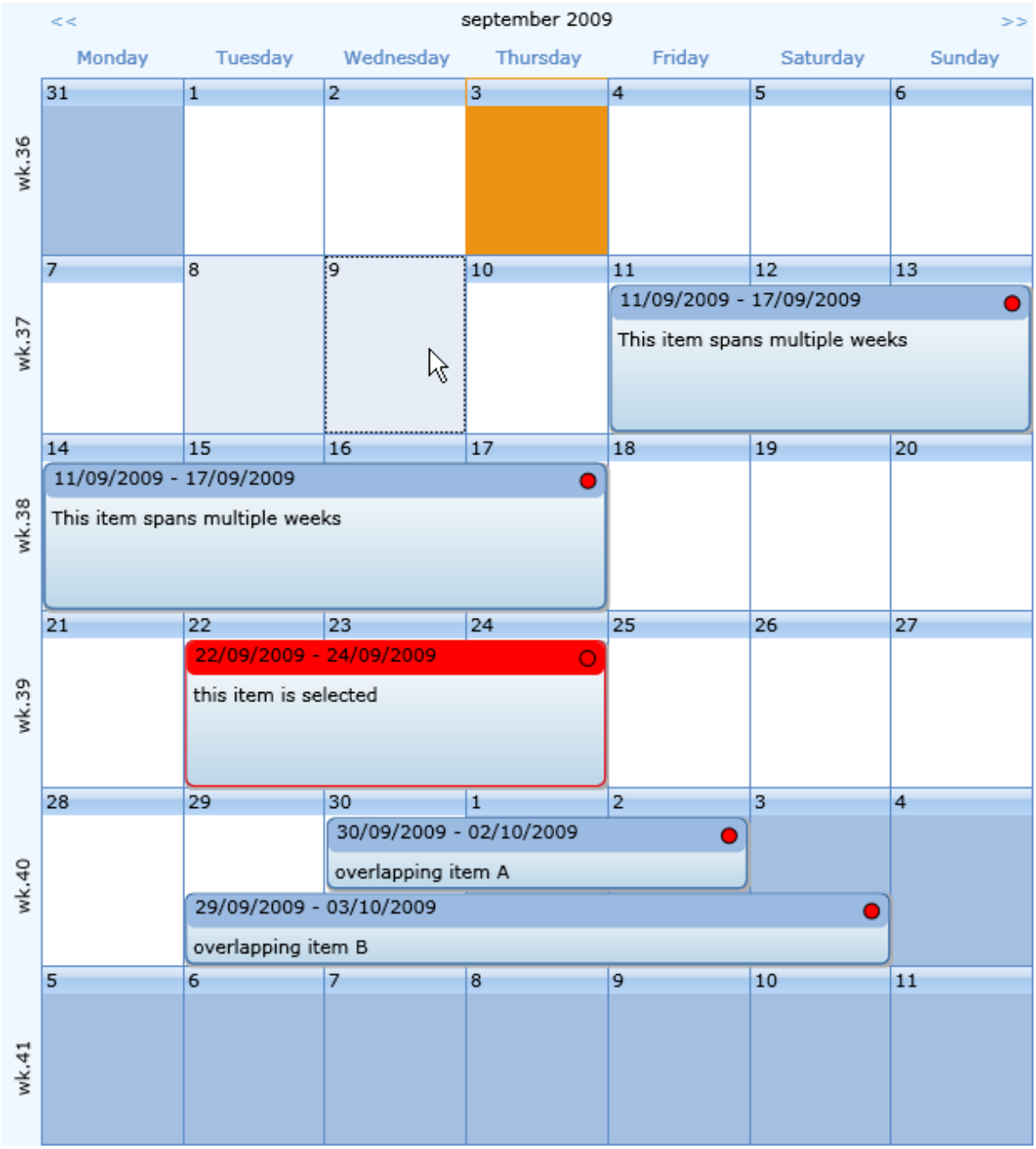
- WeekNumberTextBrush
  - Brush, defines the text brush for the WeekNumbers column.

MonthPlanner-only events overview

- GotoNextMonth
  - Fires when the NextMonth button is clicked.

- GotoPreviousMonth
  - Fires when the PreviousMonth button is clicked.

Example of a MonthPlanner with custom colors, settings and databound PlannerItems:



FAQ

- What is the correct way to edit the generic.xaml file in the Themes subfolder?
  There are a couple of small differences between the generic.xaml file that is used for the Silverlight Planner and the one that is used for the WPF Planner. Therefore a different file is copied to the Themes folder depending on which solution you are using. The file is located in a subfolder of the Themes folder called “WPF” for the WPF Planner and “Silverlight” for the Silverlight Planner.

Online references

TMS Software website
http://www.tmssoftware.com/

TMS Silverlight - WPF Planner page
http://www.tmssoftware.com/site/slplanner.asp

The Official Microsoft Silverlight site
http://www.silverlight.net/