Gantt Control Developer Reference

.NET Language Guide

VERSION 2.00 – AUG 2012

GmbH

Vordruckverlag Weise GmbH
Bamberger Str. 4-6
01187 Dresden
Germany

http://www.gantt-komponente.de/
http://www.gantt-component.com/

Table of content

This developer reference guide describes the software component GANTT CONTROL .NET for the Microsoft .NET Platform. For most of the code examples and references the software development language C# is used.

Installation

To install the GANTT CONTROL .NET component you require the **GanttControl.dll** file.

For Microsoft Visual Studio 2008, please select the “**Choose Toolbox Items…**” from the “**Tools**” Menu. After that, please activate the first tab sheet “.NET Framework Components” of the dialog that should appear now.

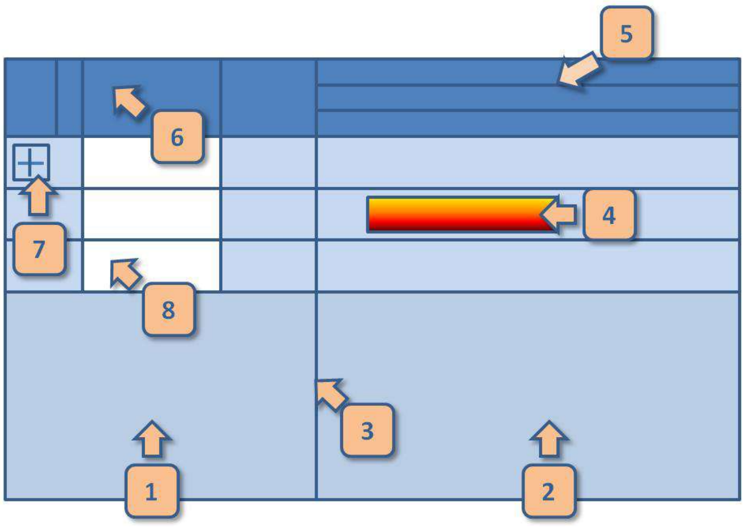
Now click on the “**Browse…**” button and select the **GanttControl.dll file**. The table below shows the components that are now available as new .NET Framework Components.

COMPONENT	NAMESPACE
GanttControl	VVW.GanttControl
PrintPreview	VVW.GanttControl.GanttPrintPreview
XmlDataSource	VVW.GanttControl.GanttDataSource

GANTT CONTROL

The Gantt Control is an interactive user interface component that contains a tree grid on the left side and a gantt chart diagram on the right side. The gantt control combines both the usability of typical grid components of having easy accessible columns and rows and also the possibility to structure the content hierarchically as it is know from common tree components.

To identify most of the elements and objects of the gantt control please see the following list.



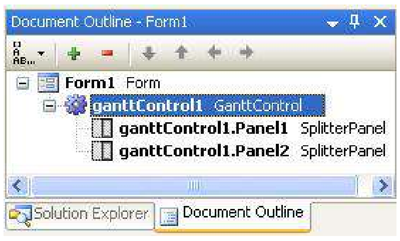
Element	Element
1	TreeGrid
2	Chart Diagram
3	Splitter
4	TaskBar
5	TimeScales[0..2]
6	Column
7	Collapsed Node
8	Tree Cell

Set up the Gantt Control Component

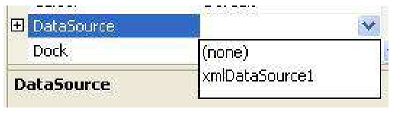
In order to set up the component for your application, please place the GanttControl component from your component tool box onto your win form. You should be able to see now an empty gantt chart diagram.

- Insert a new XMLDataSource component from the tool box on the form and connect the GanttControl component with the XMLDataSource. To do so, please select the GanttControl component.

  Please note, if you click on the component itself, it may be possible that the sub-panels (panel1 and panel2) of the GanttControl component are selected instead of the GanttControl component itself. If you have any problems, to select the GanttControl, then please activate the "Document Outline view" [CTRL + W, U] of Visual Studio.


Document Outline - Form1
Form1 Form
ganttControl1 GanttControl
ganttControl1.Panel1 SplitterPanel
ganttControl1.Panel2 SplitterPanel
Solution Explorer
Document Outline

- If the GanttControl component is selected, please open its “Properties-Dialog” [CTRL + W, P] and assign the XMLDataSource for the DataSource property.



If you have accomplished those two steps you can compile and run your .NET Windows Form Application. However the GanttControl component does not contain any rows or columns, so at the moment you are yet unable to interact with the component.

Set up new columns

By default the tree of the gantt graph does not contain any columns. When adapting the gantt control to your specific requirements its necessary to add columns to the tree. There are two different types of columns – **predefined columns** and **user defined columns**. Predefined columns are columns that are handled completely by the logic of the gantt control component itself. Predefined columns are used to display information to the end user and are not editable, as the cell content is determined and calculated by the component itself.

Predefined Column	Description
TreeAutoCellNumber	The auto cell number column displays the current number of the row. There are two different display formats for the row number. If the boolean property ***ContinuousNumbering*** of the column is set to true the rows are numbered in a continuously way, otherwise in a hierarchically way.
TreeAutoCellStartDate	The auto cell start date column displays the earliest start date of all bars for each row.
TreeAutoCellEndDate	The auto cell end date column displays the latest end date of all bars for each row.
TreeAutoCellTaskName	If a pert bar has been defined for a row the name of the pert bar is displayed in the auto cell task name column.

User defined columns are editable columns that are defined by the developer. There are different types of user defined columns, providing the editing of different data types. The following table shows all user defined columns and their description.

User defined column	Description
TreeSimpleTextCell	The simple text column is used to edit/display one single line of text.
TreeMultilineTextCell	The multiline text column is used to edit/display a multi-line text block, including word wraps.
TreeDateTimeCell	The date time column is used to edit/display date and/or time values.
TreeSpinCell	The Spin column can be used to display and edit numeric values.
TreeCurrencyCell	The currency column is used to edit and display currency values.
TreeComboCell	The combo column provides an editable drop down combo box.

TreeImageComboCell	The image combo column provides an editable combo box including an additional image.
TreeButtonEditCell	The button edit column is a composition of an editable text field and an additional button.

- The following code sample demonstrates how to add a column to the GanttControl component. The columns constructor used in this example requires the xmlDataSource, the columns type and the caption of the column as parameters. The xmlDataSource provides access to the columns array. Please add the VVW.GanttControl.GanttDataSource namespace to your project, in order to use the column constructor as shown below.

(1)

```
using VVW.GanttControl.GanttDataSource;
…
xmlDataSource1.Columns.Add(new Column(xmlDataSource1, typeof(TreeAutoCellNumber), "Nr."));
```

Based on this proceeding all other types of columns can be added to the GanttControl. Depending on the columns type, specific properties can be accessed for each column type. The example below shows how to set some column specific properties.

- After having added an auto cell number column, a combo box column is added. After that the ContinousNumbering property of the auto cell number column is set to false and some items are defined for the combo column.

(2)

```
using VVW.GanttControl.GanttDataSource;
…
…
xmlDataSource1.Columns.Add(new Column(xmlDataSource1, typeof(TreeAutoCellNumber), "Nr."));
xmlDataSource1.Columns.Add(new Column(xmlDataSource1, typeof(TreeComboCell), "Combo"));

xmlDataSource1.Columns[0].GetTreeCell<TreeAutoCellNumber>().ContinuousNumbering = false;

List<string> content = new List<string>();
content.Add("item_00");
content.Add("item_01");
content.Add("item_02");
xmlDataSource1.Columns[1].GetTreeCell<TreeComboCell>().Content = content;
xmlDataSource1.Columns[1].GetTreeCell<TreeComboCell>().DropDownStyle = ComboBoxStyle.DropDown;
…
```

The TreeEditButtonCell column provides an additional button for each cell. If you want to access the ButtonClick event of those cells, the best way would be to create a new class that inherits from the TreeComboCell class and overwrite the ButtonClick event handler.

- As explained above, we create a new class MyTreeButton and overwrite the ButtonClick event within this class.

(3)

```
public class MyTreeButton : TreeButtonEditCell
 {
     public MyTreeButton(DataSource dataSource)
         : base(dataSource)
     {
     }

     public override void ButtonClick(object sender, EventArgs e)
     {
        // Buttons on click event ...
     }
}
```

- After that, we can create the TreeButtonEditColumn by using the column constructor shown below:

(4)

```
xmlDataSource1.Columns.Add(new  Column(xmlDataSource1,  typeof(TreeButtonEditCell),  "Column",
new MyTreeButton(xmlDataSource1)));
```

Accessing single cells values

For setting and retrieving cell values the gantt control component provides the methods SetCellContent and GetCellContent.

- The sample below demonstrate how to use the SetCellContent and the GetCellContent methods.

(5)
```
ganttControl1.TreeGrid.SetCellContent(0, 1, "value");
String value = ganttControl1.TreeGrid.GetCellContent(0, 1);
```

Row operations

All data-sensitive operations (like adding a row) are performed by using the XMLDataSource component. So the XMLDataSource provides methods for adding, deleting and structuring rows. For accessing single rows the XMLDataSource provides the Rows [] array.

- For adding new rows use the **`RowAppend()`** method.

(6)
```
…
xmlDataSource1.RowAppend();
…
```

If you want to add multiple rows at once you can specify the number of rows you want to add to the GanttControl component.

- The **`RowAppend(..)`** method is used to add multiple rows at once.

(7)
```
…
xmlDataSource1.RowAppend(5);
…
```

- To delete an existing row you can use the **`RowDelete(..)`** function. The Index specifies the row index; the topmost row has an index of 0.

(8)
```
…
xmlDataSource1.Delete(1);
…
```

- If you want to delete only the content of a single row (this means deleting all cell values of a row as well as all bars that are linked to this row) you can use the **`RowClear(..)`** method.

(9)
```
…
xmlDataSource1.RowClear(1);
…
```

- If you want to insert a row at a given position you can use the **`RowInsert(..)`** function. This will insert a new empty row object at the given position.

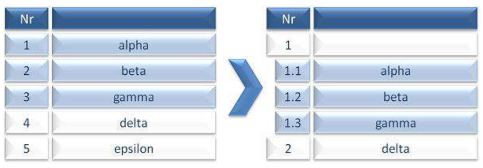
(10)
```
…
xmlDataSource1.RowInsert(1);
…
```

Hierarchical structuring of rows

The tree of the GanttControl component allows it, to create a hierarchical structure. The XMLDataSource provides different methods to apply a hierarchical tree-like structure.

- You can group a range of rows and increase their nesting level by using the RowsGroup method. The range is determined by the parameters startIndex and endIndex.

(11)
```
…
xmlDataSource1.RowsGroup(0,2);
…
```



Grouping the first three rows by using the RowsGroup(0,2) statement – the structure of the tree will alter as it is visualized in the pictogram on the left side. After grouping the first 3 rows a new row will be inserted as parent row.

Please note that it is not possible to add task bars in a parent row (also called as summary row), as the parent row calculates its duration itself and will be automatically updated, based on the duration of its child rows.

- Use the `RowChildInsert(..)` method to add a new child row for the specified row. The example below adds a new child to the first row.

(12)
```
…
xmlDataSource1.RowChildInsert(0);
…
```

- To delete all childs from a row you can use the `BranchDelete(..)` function:

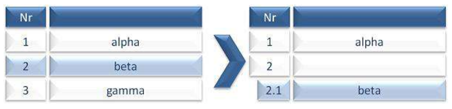
(13)
```
…
xmlDataSource1.BranchDelete(0);
…
```

- The parent row of a set of child rows can be reassigned by using the `RowsChangeParent(..)` method. The range of rows, that's parent index will be reassigned is defined by the startIndex and the endIndex.

(14)
```
…
xmlDataSource1.RowsChangeParent(1,2,0);
…
```

Basically there are two different modes how the hierarchical structure is visualized in the predefined tree auto cell number column. By default the "No"-column displays the hierarchical level as a recursive aggregation of its number and its sub-number (e.g. "1.1"; "2.1.1"). You can also apply a continuous numbering ("1", "2", …) display format by setting the `ContinuousNumbering` property to true.

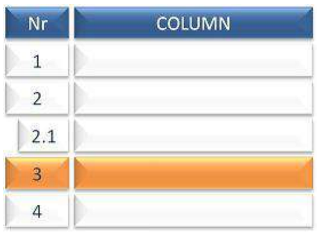
Additionally you can increase and decrease the nesting level of single rows, if you use the `IncreaseLevelOfRow` and the `DecreaseLevelOfRow` of the XMLDataSource.



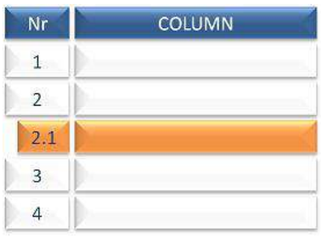
The example on the left side shows how the GanttControl changes after a `IncreaseLevelOfRow(1)` has been applied. Please note, that a new parent row has been added for the child row. If you want to reverse the changes made through the increase of the level you need to call the `DecreaseLevelOfRow(2)` method.

Whenever (child)-rows are modified (deleted or added) the value of the tree auto cell number column will be updated automatically.

When accessing rows of a tree grid that has already any child rows, you have to be carefully to use the correct row index as parameter for row operations. If you use an absolute index for accessing a row, you have to use the GetRowAtAbsoluteIndex method. The example below shows what rows are accessible by using different methods.



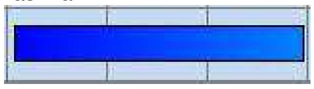
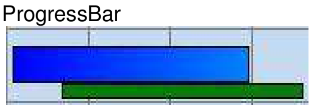
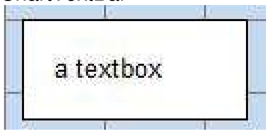
xmlDataSource1.Rows[2]



xmlDataSource1.GetRowAtAbsoluteIndex(2)

Adding bars and objects to the gantt chart

The gantt graph contains different types of bars and objects – see the following table:

Bar / Object	**Description**
TaskBar 	The taskbar is the main element of the gantt chart. It is used to visualize a single (planned/target) activity. Taskbars can be linked together in form of a single connection. Taskbars are basically defined and determined by their StartDate, their EndDate as well as the row in which they are displayed.
ProgressBar 	A progressbar is commonly used to visualize the effective progress of a task. Therefore it is displayed below a task bar in the same row as the task bar. Please note that a progressbar is not limited to be located within the range of a taskbar. Progressbars cannot be linked together with connections and they contain no internal progress status as it is the case for task bars.
ChartImage	ChartImages can visualize any image.
ChartTextBar 	Any textual information can be visualized by using the ChartTextBar.

The end user is able to create new bars, if he holds down the left mouse button and drags a new object into a row of the gantt chart. By default all new bars that are created this way (by the user) are task bars.

If you set the `TaskMode` property of the Graph to false (`ganttControl1.Graph.TaskMode = false;`) the component will create progress bars instead of task bars, as described above.

Use the methods `AddTaskBar` and `AddProgressBar` if you want to add task bars and progress bars via code.

- The sample below adds a new task bar in the first row. The task bar starts on the current day and has a total duration of 3 days. After that a progress bar is added directly below the task bar in the same row. The progress bar lasts one day longer than the task bar. Please note in a typical task scheduling context the task bar always represent a planned activity as the progress bar visualizes the actual progress of an activity. In this short example the task has a total delay of one day.

(15)
```
…
xmlDataSource1.Rows[0].AddTaskBar(new TaskBar(xmlDataSource1, DateTime.Now,
DateTime.Now.AddDays(3)));

xmlDataSource1.Rows[0].AddProgressBar(new ChartProgressBar(xmlDataSource1, DateTime.Now,
DateTime.Now.AddDays(4)));
…
```

If you want to add **`ChartImage`** and **`ChartTextBar`** objects you have to add those objects to the **`xmlDataSource.ChartObjects`** array. The example below shows how this could be done.

- Use the **`Add`** method of the **`ChartObjects`** class to add **`ChartImages`** and **`ChartTextBars`**.

(16)
```
…
int selectedLine = ganttControl1.Graph.SelectedLine;

ChartImage chi = new ChartImage(xmlDataSource1, ganttControl1.Graph.StartingDate.AddDays(3),
xmlDataSource1.Rows.RowAtAbsoluteIndex(selectedLine));
chi.EndDate = ganttControl1.Graph.StartingDate.AddDays(6);
chi.AutoStretch = false;

chi.ChartGraphic = pictureBox3.Image; //reference to an image object
chi.Tag = 0;
xmlDataSource1.ChartObjects.Add(chi);
…
```

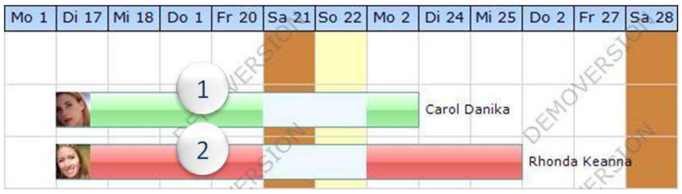
Task bars

As shown above, the GanttGraph provides four bar types. The most important – as this is the basic bar of gantt diagrams – is the taskbar. This chapter concentrates mainly on the properties of the task bars and the possibility to change the layout of a task bar. This chapter concentrates mainly on the properties of the task bars and the possibilities to change the layout of a task bar. If you want to know how to add a task bar or how to access a task bar – please have a look at the prior chapter.

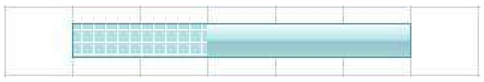
A task bar is defined by its start and end date. Use the **`StartDate`** or **`EndDate`** property to assign a new value or to read out the current value. Please note that when assigning a new time range for task bars the **`EndDate`** must always been larger than the **`StartDate`**. The Duration of a task bar can be obtained by read-out the **`Duration`** property of task bars.

In most cases you probably have defined some non working dates in the underlying calendar of the GanttGraph, respectively they are already defined by default, e.g. weekends. If so – you can directly get the time, the task bar fits to working times using the **`EffectiveDuration`** property.

To demonstrate the interaction between non-working times and effective duration, please have a look at the following graphic. Here the weekends are defined as non working times, therefore the second bar (2) has an effective duration of 7 days and the first bar (1) has an effective duration of 5 days.



If you want to highlight the beginning and/or the ending of a bar you can use buffer times. In its original meaning buffer times have been implemented to visualize reserved time ranges, where no other task can occupy the time of the buffer time, for a given task bar. To clarify this, please imagine the following scenario, where the usage of a buffer time may be wise. You want to schedule the allocation of different machines. Every time you allocate a new machine, the machine needs some reserved time to be built up and to be adjusted. So one way to visualize a machine that has an initial "setup time" of 2 days and a "working time" of 3 days can be done as shown in the following picture, where the bar has a left buffer time of two days and an overall duration of 5 days.



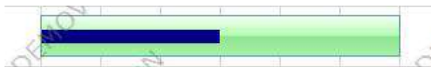
Buffer times are visualized using a pattern. If you want to assign a buffer time you can use the both properties DateTime `BufferLeftTime` and `BufferRightTime`. There are some restrictions that you have to keep in mind when using BufferTimes:

- `BufferLeftTime` should always be smaller than `BufferRightTime`
- `BufferLeftTime` and `BufferRightTime` should always be larger than the `StartDate` and smaller than the `EndDate` of the task bar.

The following table summarizes the described properties concerning the time scheduling of a task bar as well as buffer times.

Property	Description
DateTime StartDate	The start date of a task bar.
DateTime EndDate	The end date of a task bar
int Duration	(read only) The overall duration of the task bar.
int EffectiveDuration	The duration of working time, based upon the calendar settings, of the task bar.
DateTime BufferLeftTime	The buffer time for the start of the task bar
DateTime BufferRightTime	The buffer time for the end of the task bar.

For each task that is visualized by a task bar, you are able to assign and display the progress of this task, within a range from 0% to 100%. The progress of the task bar is displayed as a small bar within the task bar. The picture below shows a task bar where the progress was set to 50 percent.



If you want to assign the internal task bar progress use the `InternalProgress` property with a value range from 0 to 100.

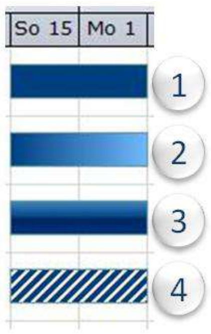
In most cases it can be useful to link further textual information to a task bar. For this purpose task bars (like other bar objects) provide the `InfoLabel` property. You can assign five text strings to a task bar, differing in their position. The following table gives a brief summary of all properties.

Property	Description
string InfoLabel.LeftText	A text – on the left side of a task bar.
string InfoLabel.TopText	A text – on the top side of a task bar.
string InfoLabel.RightText	A text – on the right side of a task bar.
string InfoLabel.BottomText	A text displayed below the task bar.
string InfoLabel.CenterText	A text displayed inside the task bar.

If you want to adjust the layout of the text of a taskbar you have to use the `InfoLabel.TextFont` property that provides parameters for adjusting the font of the text.

In the rest of this chapter we will focus more on how to adjust the visual appearance of task bars.

There are different styles that can be applied to draw a task bar. First there is a solid fill style (1) a gradient fill style (2) a three dimensional drawing style (3) and a draw style using a pattern (4).



If you want to change the draw style of the task bar please assign a value shown in the following table to the `Style` property.

Style	Description
(1) Solid	Solid fill style
(2) Gradient	Gradient (vertical or horizontal) fill style
(3) D3Look	Three dimensional draw style
(4) Pattern	Pattern draw style.

According to the draw style that you have selected for the task bar, there are different properties that are used when the task bar is drawn.

Solid or D3Look Style	
Property	**Description**
`BackgroundColor`	Specifies the background color of the bar

Gradient Style	
Property	**Description**
`GradientStartColor`	Specifies the start color of the gradient fill
`GradientEndColor`	Specifes the end color of the gradient fill
`GradientStyle`	Specifies the gradient mode (BackwardDiagonal, ForwardDiagonal, Horizontal, Vertical)

Pattern Style	
Property	**Description**
`BackgroundColor`	Specifies the background color of the pattern.
`PatternColor`	Specifies the foreground color of the pattern.
`PatternStyle`	The style of the pattern.

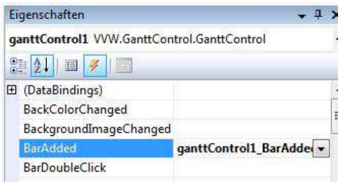
When the (end)user adds a new task bar to the gantt graph or a task bar is added by program logic, its layout parameters are defined by default settings. You can override any of these settings to specify the default layout for task bars, so that it fits the needs of your application. Please see the following list for all parameters that specify the default layout of task bars.

Property	Description
DefaultTaskBarBackgroundColor	Default background color of the bar
DefaultTaskBarBackgroundNonWorkingColor	Default color when the bar is over non working time
DefaultTaskBarBorderColor	Default border color of the task bar
DefaultTaskBarBorderThickness	Default border thickness
DefaultTaskBarGradientStartColor	Default gradient start color for gradient style bars
DefaultTaskBarGradientEndColor	Default gradient end color for gradient style bars

DefaultTaskBarGradientStyle	Default gradient style
DefaultTaskBarHeight	Default height of a task bar
DefaultTaskBarPatternColor	Default pattern foreground color of task bars
DefaultTaskBarPatternStyle	Default pattern style
DefaultTaskBarProgressColor	Default color of the InternalProgress
DefaultTaskBarStyle	Default drawing style of task bars.

Another way to adapt the layout of task bars is, to use the **`BarAdded(object sender)`** event of the GanttControl component.

- After adding a new bar, the **`BarAdded`** event of the GanttControl component will be raised. First we test here, if the added bar is a task bar – if so, we will change some layout parameters of the task bar. To edit the **`BarAdded`** event handler, please select the GanttControl component, select the “events” tab page and double click on the Bar Added event.

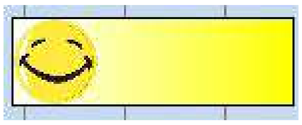


Eigenschaften

ganttControl1 VVW.GanttControl.GanttControl

(DataBindings)	
BackColorChanged	
BackgroundImageChanged	
BarAdded	ganttControl1_BarAdded
BarDoubleClick	

(17)
```
…
private void ganttControl1_BarAdded(object sender)
 {
  if (sender.GetType() == typeof(TaskBar))
   {
     ((TaskBar)(sender)).Height = 33;
     ((TaskBar)(sender)).Style = TaskBarStyle.Gradient;
     ((TaskBar)(sender)).GradientStartColor = Color.Yellow;
     ((TaskBar)(sender)).GradientEndColor = Color.Red;
     ((TaskBar)(sender)).InfoLabel.RightText = "task bar";
   }
 }
…
```



After adding text and changing the layout of a task bar, it is shown now how to add images to a task bar. Each task bar can own a small glyph that will be displayed inside the task bar at the start (**`StartGlyph`**) and/or at the end (**`EndGlyph`**) of the bar.

- The code sample below extends the **`BarAdded`** event and adds a **`StartGlyph`** to the bar that is loaded from a file at runtime.

(18)
```
…
private void ganttControl1_BarAdded(object sender)
 {
   if (sender.GetType() == typeof(TaskBar))
    {
      ((TaskBar)(sender)).Height = 33;
      ((TaskBar)(sender)).Style = TaskBarStyle.Gradient;
      ((TaskBar)(sender)).GradientStartColor = Color.Yellow;
      ((TaskBar)(sender)).GradientEndColor = Color.Red;
      ((TaskBar)(sender)).StartGlyph = Image.FromFile("D:/startglyph.bmp");
    }
 }
…
```

Besides the properties that have been mentioned already, the task bar provides some other properties, as listed in the table below:

Property	Description
Color BackgroundNonWorkingColor	The color that is used when the bar is placed over non working time.
Color BorderColor	The color value for the border of the bar.
int BorderThickness	The thickness in number of pixels that is applied for the bars border.
bool CriticalElement	Returns true if the bar is part of the critical path
int Height	Specifies the total height in pixel of the task bar
Row ParentRow	Returns the row object that belongs to the task bar

Progress bars

Progress bars can be used to visualize the actual progress a task has made. For a better comparison they are directly drawn under the task bar, so they may be applicable for any kind of target-performance comparisons, as the user can see the planned time for an activity and the actual time the task has needed.

When creating a progress bar you are not restricted in any way – you can create as many progress bars for each row as you want. The amount of progress bars does not depend on the amount of task bars, nor is a progress bar in anyway linked to a task bar. If you want to know how to add a progress bar or how to access a progress bar – please have a look at the prior chapter.

As well as the task bar, the progress bar is defined by its **`StartDate`** and its **`EndDate`**. Use the **`StartDate`** and **`EndDate`** properties to define the length and the position of the progress bar.

Property	Description
DateTime StartDate	The start date of a progress bar
DateTime EndDate	The end date of a progress bar

As it has been shown for task bars, progress bars can display textual information to. You can assign five text strings to a progress bar, differing in their position. The following table gives a summary of all properties.

Property	Description
string InfoLabel.LeftText	A text – on the left side of a progress bar
string InfoLabel.TopText	A text – on the top side of a progress bar
string InfoLabel.RightText	A text – on the right side of a progress bar
string InfoLabel.BottomText	A text displayed below the progress bar
string InfoLabel.CenterText	A text displayed inside the progress bar.

If you want to adjust the layout of the text of a progress bar, you have to use the **`InfoLabel.TextFont`** property that provides parameters for adjusting the font of the text.

For the rest of this chapter we will focus on properties that define the layout of the progress bar. In comparison to task bars, progress bars do only have one solid fill draw style implemented. If you want to change the color of a progress bar use the **`BackgroundColor`** property.

The other relevant properties for progress bars are shown in the table below:

Property	Description
int Height	The height of a progress bar in pixel.

Color BorderColor	The color value of the border of the progress bar.
int BorderThickness	The value of the thickness of the border in pixel.

- The following code example shows how to add a progress bar and change some of its properties. Please note that the code for adding and formatting progress bars is placed between a **`BeginUpdate()`** and **`EndUpdate()`** statement.

  Whenever you plan to manipulate a lot of data, e.g. adding a lot of rows, its recommend to place the operation(s) between a **`BeginUpdate()`** and an **`EndUpdate()`** statement to enhance the performance of your software application. The second code of line scrolls the visible range of the gantt chart to the current date. So that the added progress bar became visible.

(19)
```
…

  xmlDataSource1.BeginUpdate();
  ganttControl1.Graph.Calendar.SetStartingDate(DateTime.Now);
  xmlDataSource1.Rows[0].AddProgressBar(new ChartProgressBar(xmlDataSource1, DateTime.Now,
DateTime.Now.AddDays(4)));

  xmlDataSource1.Rows[0].ProgressBars[0].InfoLabel.RightText = "progress bar";
  xmlDataSource1.Rows[0].ProgressBars[0].BackgroundColor = Color.Yellow;
  xmlDataSource1.Rows[0].ProgressBars[0].Height = 20;
  xmlDataSource1.Rows[0].ProgressBars[0].BorderColor = Color.Red;
  xmlDataSource1.Rows[0].ProgressBars[0].BorderThickness = 2;
  xmlDataSource1.EndUpdate();

…
```

Milestones

Milestones represent a special date in your project, so for example a specific deadline. They are visualized as a small glyph.

- The following code example shows how to add a milestone to the GanttControl component and assign an image to it.

(20)
```
…
ChartMilestone chms = new ChartMilestone(xmlDataSource1,
                          DateTime.Now, xmlDataSource1.Rows[1]);
chms.ChartGraphic = Image.FromFile("D:/milestone.bmp");
xmlDataSource1.ChartObjects.Add(chms);
…
```

As already known from the other bars the chart milestone object has also the **`InfoLabel`** property.

Property	Description
string InfoLabel.LeftText	A text – on the left side of a milestone chart object
string InfoLabel.TopText	A text – on the top side of a milestone chart object
string InfoLabel.RightText	A text – on the right side of a milestone chart object
string InfoLabel.BottomText	A text displayed below the milestone chart object.
string InfoLabel.CenterText	A text displayed inside the milestone chart object.

Images

If you want to display a graphic within the GanttControl component you can add a ChartImage object to the XMLDataSource. Please keep in mind that you are unable to display images (and milestones too) within the left part (tree) of the GanttControl component. Please have a look at the following code snippet, to see how a ChartImage is added to the gantt chart.

If you want to display a graphic within the GanttControl component you can add a ChartImage object to the XMLDataSource.

- The following code example shows how to add a chart image to the GanttControl component and assign an image to it.

(21)
```
…
ChartImage chi = new ChartImage(xmlDataSource1, DateTime.Now, xmlDataSource1.Rows[1]);
chi.ChartGraphic = Image.FromFile("D:/chart_image.bmp");
xmlDataSource1.ChartObjects.Add(chi);
…
```

ChartImage objects do have – as all other types of bars – the **`InfoLabel`** property, allowing them to display some text inside/beside the image.

Property	Description
string InfoLabel.LeftText	A text – on the left side of a image chart object
string InfoLabel.TopText	A text – on the top side of a image chart object
string InfoLabel.RightText	A text – on the right side of a image chart object
string InfoLabel.BottomText	A text displayed below the image chart object.
string InfoLabel.CenterText	A text displayed inside the image chart object.

Text bars

Text bars can be used to display larger text information within the GanttGraph. A Chart text bar as well as image bars can cover more than one row of the GanttGraph. When creating a chart text bar object its horizontal dimensions are defined by the **`StartDate`** and the **`EndDate`**. To define the vertical dimension you have to use the Size property of the Chart text bar that specifies the height of the text bar in pixel. To assign some text to the text bar use the **`Text`** property.

The following code example demonstrates how to create a text bar.

- The following code example shows how to add a chart image to the GanttControl component and assign an image to it.

(22)
```
…
ChartTextBar chtb = new ChartTextBar(xmlDataSource1, DateTime.Now, xmlDataSource1.Rows[1]);
chtb.Text = "text of a text bar";
chtb.EndDate = chtb.StartDate.AddDays(6);
xmlDataSource1.ChartObjects.Add(chtb);
…
```

The following table shows further properties of text bars:

Property	Description
`Font TextFont`	Specifies the font of the text bar
`Color BackgroundColor`	The background color of the text bar. The color is applied only when Transparent is set to false.
`StringAlignment HAlignment`	The horizontal alignment of the text. The following sub-table shows the possible values and their corresponding alignments. *Value* / *Alignment*: Near – Left; Far – Right; Center – Center
`StringAlignment VAlignment`	The vertical alignment of the text. The following sub-table shows the possible values and their corresponding alignment. *Value* / *Alignment*: Near – Top; Far – Bottom; Center – Center
`bool Transparent`	The text bar will be displayed transparent if set to true.

HAlignment sub-table:

Value	*Alignment*
Near	Left
Far	Right
Center	Center

VAlignment sub-table:

Value	*Alignment*
Near	Top
Far	Bottom
Center	Center

Time scales and time mode

The gantt chart diagram provides three time scale objects, each with a different time mode. The time mode of the time scales is defined by the global time mode of the calendar. Use the `SetTimeMode(Calendar.TimeModes timeMode)` property of the Graph to change the time mode. The following table shows all possible values for the time mode:

Value	*Scheduling mode*
Hour	Hourly view mode
Day	Daily view mode
Week	Weekly view mode
Month	View mode for months
Quarter	View mode for quarters
Year	View mode for years
Decade	View mode for decades

- The next line of code changes the global time mode of the calendar to a weekly mode.

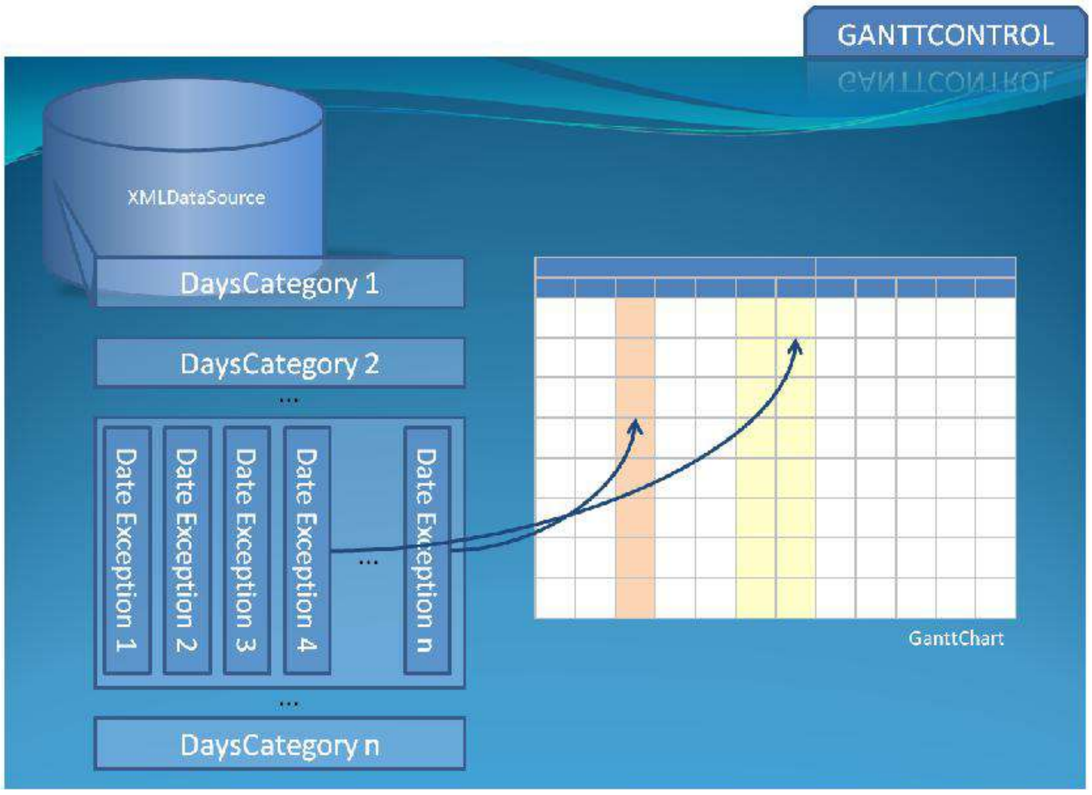
(22)
```
…
ganttControl1.Graph.SetTimeMode(Calendar.TimeModes.Week);
…
```

Please note, the content of date specific strings, such as day names or month names is based on the global language settings of the system.

Working and non working times

The GanttControl component contains the logic of an internal calendar that can be used to define and visualize special (working or non working) date exceptions. However when you define dates within the calendar, they are applied to the overall Gantt Graph. Please note, that it is not possible to define different date exceptions for different rows.

Please have a look at the following picture that symbolizes the architecture of the Calendar.



As you can see the XMLDataSource holds a list of date categories `xmlDataSource1.DaysCategories`. For each DaysCategory special dates can be defined.

The appearance and the behaviour of a single date exception is defined by its DaysCategory. The DaysCategory specifies whether its date exception defines working or non working times. Also the visibility and the color that is used to visualize the date exception can be specified within the date category. The following table summarizes the properties of a date category.

Property	Description
Brush BrushFormat	The brush that is used when the date exceptions of this DaysCategory are drawn.
int DaysCount	Returns the total number of date exceptions that are defined within this DayCategory
string Name	The name of the DaysCategory. Note: This name is just for the purpose to identify the DayCategory.
bool Visible	Specifies whether date exceptions of this DaysCategory are displayed or not.
bool Working	Specifies whether date exceptions of this DaysCategory should be handled as working dates/times or not.

Additionally the DaysCategory holds several methods for managing the date exceptions of a DaysCategory:

Method	Description
`void Clear()`	Clears all date exceptions of the DaysCategory.
`bool ContainDate(DateTime date)`	Returns true if the DaysCategory contains an entry for the specified date
`void RemoveDate(DaysCategory.DayCategory dayCategory)`	Removes the date exceptions entry from the DaysCategory.

If you want to add new dates to a DaysCategory the component provides the method AddDate(…) that can be called using different parameter lists.

```
void AddDate(DateTime date);
```

- This method adds a DateException, defined by its DateTime value, to the DaysCategory.

```
void AddDate(DaysCategory.DayCategory dayCategory);
```

- This method adds a DateException, defined by a DayCategory object, to the DaysCategory.

```
void AddDate(DateTime date, Calendar.TimeModes repeatMode);
```

- This method is used to add recurring DateExceptions, defined by a DateTime value and the repeatMode to the DaysCategory. Please see the following list for the possible values for repeatMode:

Calendar.TimeModes
Hour
Day
Week
Month
Quarter
Year
Decade
None

```
void AddDate(DateTime date, Calendar.TimeModes repeatMode, int repeatAfter);
```

- This method is used to add recurring DateExceptions, defined by the DateTime value and the repeatMode. The repeatAfter parameter specifies the time the DateException is repeated after. For example; if the repeatAfter is set to 2, the DateException will repeat after 2 times the repeatMode.

```
void AddDate(DateTime date, Calendar.TimeModes repeatMode, int repeatAfter,
DateTime rangeStart, DateTime rangeEnd);
```

- This method is used to add recurring DateExceptions, as described above. Additionally you can set up a time range. So that only valid DateExceptions that are inside the time range are created.

If there are date exceptions in different DaysCategories that overlap or share the same date, it is important to consider the order of the categories in the categories list. The date exceptions of the category that is the last in the category list will be displayed topmost within the gantt chart. Generally you should try to avoid assigning the same date to different categories, as it is possible that there are contrary properties (working and non-working) for each category.

The DaysCategory list can be accessed throughout the XMLDataSource. The object holds several useful methods for adding, deleting or changing the priority of DaysCategories as shown in the table below:

Method	Description
`void Add(DaysCategory dayCategory)`	Adds a new DaysCategory object to the list.
`void Clear()`	Clears the list of DaysCategories.
`DaysCategory DaysCategoryOfDay(DateTime day)`	Returns the Dayscategory that fits to a specific day
`void MoveDown(string name)`	Moves the DaysCategory, specified by the name, one position down.
`void MoveUp(string name)`	Moves the DaysCategory, specified by the name, on position up.
`void Remove(DaysCategory dayCategory)`	Removes the specified DaysCategory.
`void Remove(int index)`	Removes the specified DaysCategory.
`void Remove(string name)`	Removes the specified DaysCategory.

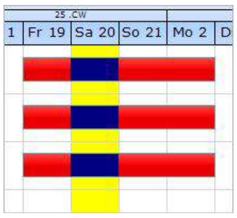
- In the following code example, we will show how to add a new date category and how to add a fixed date exception to it. A fixed date exception would be the first choice to visualize a single date/day. If you want to define recurring date exceptions please have a look at the next code example.

(23)
```
…
DaysCategory fixedException = new DaysCategory(xmlDataSource1, "non working day", false);
fixedException.BrushFormat = new SolidBrush(Color.Yellow);
fixedException.AddDate(new DateTime(2009, 6, 20));
xmlDataSource1.DaysCategories.Add(fixedException);

ganttControl1.Graph.Calendar.SetStartingDate(new DateTime(2009, 6, 20));
…
```

- The code example below shows, how recurring (repeating) date exceptions can be added. As we use another version of the AddDate(..) function as above, we can specify that the recurring time period for the date exception is saturday every second week.

(24)
```
…
DaysCategory fixedException = new DaysCategory(xmlDataSource1, "saturday", false);
fixedException.BrushFormat = new SolidBrush(Color.Yellow);
fixedException.AddDate(new DateTime(2009, 6, 20),
                           VVW.GanttControl.GanttGraph.Calendar.TimeModes.Week, 2);
xmlDataSource1.DaysCategories.Add(fixedException);

ganttControl1.Graph.Calendar.SetStartingDate(new DateTime(2009, 6, 20));
…
```

As you may have noticed, as soon as a task bar is over a non working date exception, the GanttControl component uses a different color to visualize that there is a non working area within the task bar. You can use the task bar property **BackgroundNonWorkingColor** to specify the color that should be used for drawing this/these day(s).

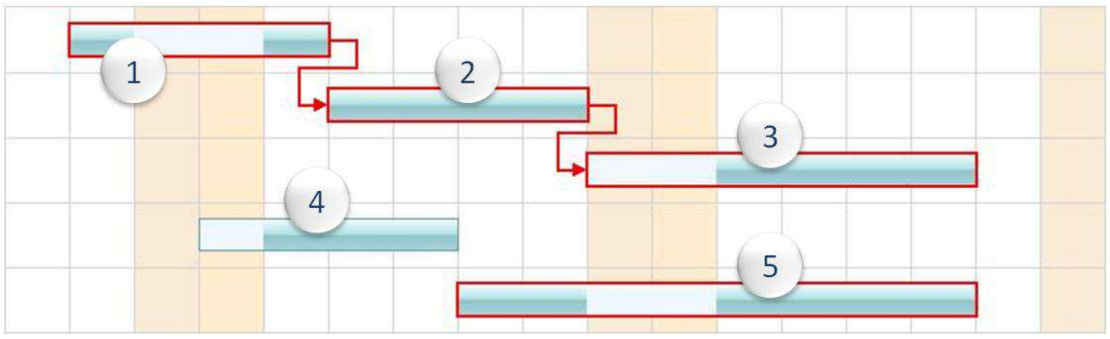


Critical Path

The Gantt Control component includes the calculation of the critical path. The calculation of the critical path determines which tasks are critical and non-critical. When a task is critical any delay of this task will result in a delay of the entire project. The critical path is the sequence of all critical tasks. Note that there can be more critical paths than one.

A typical critical path contains usually a set of connected bars. As soon as the first task, that is represented by the first bar, is delayed the second connected bar will be rescheduled, finally resulting in the rescheduling of the project end date.

The following image displays a gantt chart that contains two critical paths (1,2,3) and (5) – visualized by a red frame. In this example you can see that the only bar that will not reschedule the project end is bar (4) and therefore it is not part of the critical path.



The visualization and the calculation of the critical path can be enabled/disabled if you use the **bool UpdateCriticalPath** property of the graph. The color that is used for highlighting the critical path can be specified if you use the **Color CriticalPathColor** property of the graph.

Property	Description
`bool ganttControl1.Graph.UpdateCriticalPath`	Enables or disables the calculation of the critical path
`Color ganttControl1.Graph.CriticalPathColor`	Specifies the color that is used to draw the critical path

By default the critical path is not enabled. However please note that if the critical path is enabled and a new bar is added the bar became automatically part of the critical path (and will be highlighted therefore) if it is the last bar of the project, as the last bar is interpreted as the project end. This may irritate the end user if he is not aware of the fact that the critical path is displayed within the gantt chart diagram.

To decide whether an object is part of the critical path or not you can access the boolean property **bool CriticalElement** of a task bar. The property is read- and writeable, although it is recommended to not change the value as it is calculated internally by the gantt component itself.

Property	Description
`DateTime ESTime`	A DateTime property defining the earliest possible start time of a task/bar. The earliest starttime depends on the connection ending at the bar. The task cannot start earlier because other tasks have to be finished / started first.
`DateTime LFTime`	A DateTime property defining the latest possible finish time of a pert bar. The latest possible finish time of a task is calculated by the latest possible start time and its duration.
`double LBTime`	A double property defining the free buffer of a task. The free buffer is the number of intermediate days between this bar and the earliest connected followed bar.
`double GBTime`	A double property defining the global puffer of a task. The global buffer is the number of days that a task can delay without influencing the final project end time. If the global buffer is 0 then any delay or this bar / task would extend the overall project time. If the global buffer is zero, then the according task / bar is part of the critical path.
`DateTime EFTime`	A DateTime property defining the earliest finish time of a task. The earliest possible finish time of a task is calculated by the earliest possible starttime and its duration.
`DateTime LSTime`	A DateTime property defining the latest possible start time of a task bar. The latest start time depends on the connection starting from the bar. If the bar would be moved only one day later then the whole project ending date would be delayed.

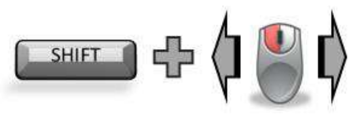
Those dates can be accessed throughout the row object.

- In the next code sample we will show how you can access the time scheduling specific properties, listed in the table above. Please not that the dates shown above are read only values. The following code example assigns the earliest start of the second row to the variable earliestStart.

(25)
```
…
 DateTime earliestStart = xmlDataSource1.Rows[1].ESTime;
…
```

Connection between bars

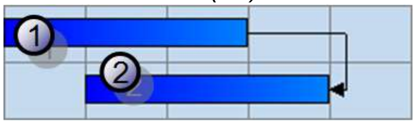
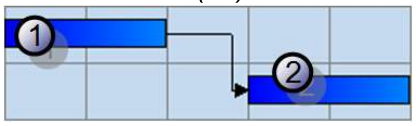
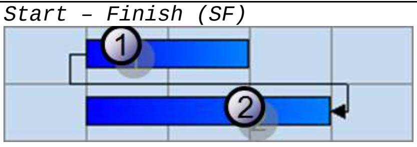
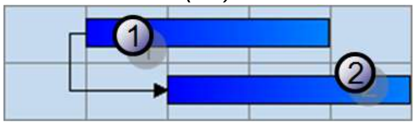
For modeling temporal and causal relationships between two tasks / activities you can connect two bars with each other and create a connection. Connections are represented as `Connection` objects.



There are two different ways to create a connection between two bars. First you can create a connection at runtime via user interface interaction. To do so, please move the mouse cursor over the start or the end of a task bar, hold down the [**Shift**] key and press the left mouse button down, move the mouse cursor to the start or the end of another taskbar and release the mouse button. After that a new connection will be established between those two bars. If necessary the connected bars will be rescheduled according to the type and the causal relationship of the connection.

Also you can create a `Connection` at runtime, to do so you have to create a Connection object and add it to the `Connections` list.

Basically there are four different types of connections, describing different causal relationships between two bars.

Connection (Example)	Restriction	ConnectionType
Finish – Finish (FF) 	Task (#2) ends always after the end of task (#1).	*FinishFinish*
Finish – Start (FS) 	Task (#2) starts always after the end of task (#1).	*FinishStart*
Start – Finish (SF) 	Task (#2) ends always after the start of task (#1)	*StartFinish*
Start – Start (SS) 	Task (#2) starts always after the start of task (#1)	*StartStart*

- The following code example creates a new finish-start connection between the first bar in the first row and the first bar in the second row and adds it to the list of connections. If you want to reproduce this code example, you have to make sure that the rows and their bars exist. After we have created and added the connection, we reschedule the bars based on the type of the connection, by calling the **`ResolveConnectionDistanceViolations`** statement.

(25)

```
…
 xmlDataSource1.BeginUpdate();
 Connection xConn = new Connection(xmlDataSource1, Connection.ConnectionType.FinishStart,
                                   xmlDataSource1.Rows[0].TaskBars[0],  // start bar
                                   xmlDataSource1.Rows[1].TaskBars[0],  // end bar
                                   0);                                   // minimal distance
 xmlDataSource1.Connections.Add(xConn);
 xmlDataSource1.ResolveConnectionDistanceViolations(xConn, true);
 xmlDataSource1.EndUpdate();
…
```

Besides the type of the connection, the connection holds other properties, which are listed in the table below:

Property	Description
`ConnectionType ConnType`	Specifies the type of the connection. See the table above.
`bool CriticalElement`	Returns true if the connection is part of the critical path
`bool DistanceFixed`	Specifies whether the distance between the two connected bars should stay fix when scheduling one of the connected bars.
`double Duration`	The distance between two bars.
`int MinDistance`	The minimal distance between the two connected Bars in days.
`TaskBar Origin`	The task bar where the connection starts from.
`TaskBar Target`	The task bar where the connection ends.

DataSource (xmlDataSource)

The DataSource component provides methods and functionalities to modify the data of the GanttControl component. The GanttControl and the PrintPreview component must be linked to the DataSource. The DataSource also holds an UndoRedoManager that makes it possible to undo/redo certain operations.

The following table shows a list of all operations provided by the DataSource component:

Property	Description
`BeginUpdate`	Call this statement, if you intend to make a lot of changes to the GanttControl component, e.g. adding bars, rows. Do not forget to call the EndUpdate statement after that.
`BranchDelete`	Removes a row, including all childs rows from the GanttControl component.
`ClearAll`	Clears the entire GanttControl component. There is a boolean parameter that makes it possible to prevent the columns from deleting.
`DecreaseLevelOfRow`	Decreases the nesting level of a row.
`EffectiveDaysInBar`	Returns the effective (working) days of a task bar.
`EndUpdate`	Finally update the GanttControl component. A BeginUpdate statement should be called before.
`FilePath`	Here you should specify the xml file that is used for loading and saving.
`GetAbsoluteIndexOfRow`	Returns the absolute index of a row.
`GetRowAtAbsoluteIndex`	Returns the Row object for the given absolute index.
`GetRowsInterval`	Returns a set of rows, specified by the startindex and the endindex of the row.
`IncreaseLevelOfRow`	Increases the nesting level of a row.
`Load`	Loads the content of the GanttControl from an xml file.
`ReNumberRows`	Renumbers the cell values for the TreeAutoCellNumber columns. Normally this function is uses only internally be the component.
`ResolveConnectionDistanceViolations`	Reschedule the bars, based on the restrictions of the connection.
`RowAppend`	Adds one (or more) rows to the GanttControl component.
`RowChildInsert`	Adds a new child row to a specified parent row.
`RowClear`	Clears the content (bars, cell values, ...) of a single row object.
`RowDelete`	Removes a row.
`RowInsert`	Inserts a new row at a specified index position.
`RowsChangeParent`	Changes the parent row for a set of child rows.
`RowsGroup`	Groups a set of rows.
`Save`	Saves the content of the GanttControl component to an xml file.
`TotalCountOfRows`	Returns the total number of rows.

The DataSource holds references to other classes. See the following table for an overview of classes that are accessible.

Property	Description
`ChartObjects`	All chart images, chart text bars are stored in this collection.

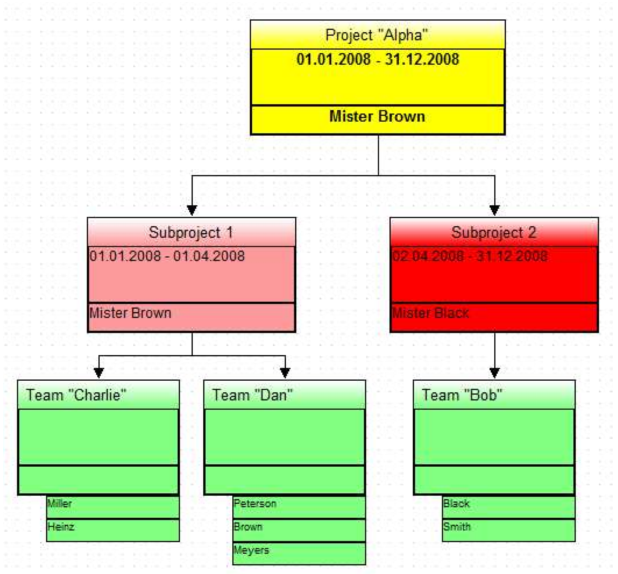
`Columns`	A collection that holds all columns.
`Connections`	A collection that holds all connections.
`DaysCategories`	A collection that holds all DaysCategories.
`Rows`	A collection that holds all Row objects

Pert chart diagram

The Pert graph diagram is another view mode that can be used instead of the gantt chart diagram. Unlike the gantt chart the pert chart visualizes single tasks as pert bars. There can be a maximum of one pert bar for each row. The pert chart area does neither have a time scale nor is it structured in rows. Two pert bars can be connected to each other.

As the pert graph is just another view mode for the same data, all data manipulating actions (adding/deleting rows, bars) are reflected in both view modes. For example; if someone deletes a pert bar in the pert chart diagram, the component automatically deletes the corresponding task bars for the gantt chart diagram.

To model existing structures of a project, pert bars can be used too. Each pert bar is divided into three segments and an optional list displaying additional items. Note: You can use the pert chart for structuring and segmenting projects. Higher level projects aims, sub ordinate project aims and work packages can be visualized and modeled as elements of the pert chart. For example pert bars that represent work packages can be extended by a list of responsible employees. For modeling relations between aims and working packages the pert chart supports connecting pert bars with each other. Please have a look at the following picture for an example of hierarchical project structuring using pert bars.

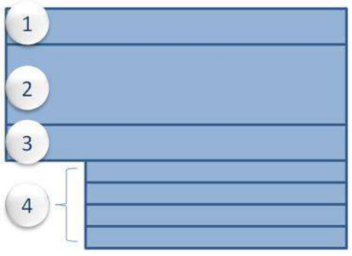


Note: All pert bars shown (referred as type I) in the example above do not display any critical time requirements. Therefore the **`TypeTimeSpan`** property of the pert bars is set to **`false`**.

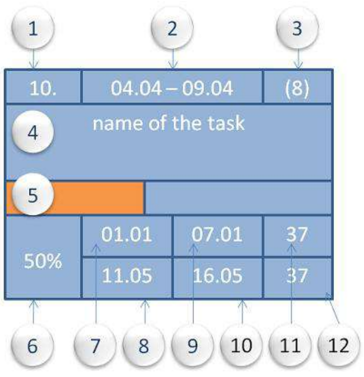
If you intend to use the pert bars for displaying critical time requirements, as the earliest start date, the earliest end date, the latest start date and the latest end date of a task, you have to set the **`TypeTimeSpan`** property to true.

Pert bar (Type I – TypeTimeSpan = false)

The pert bar type I is divided into four different segments. These are (1) TopSection, (2) MainSection, (3) BottomSection and (4) AdditionalSections.



Pert bar (Type II – TypeTimeSpan = true)



The pert bar type II displays additional time span information as shown in the picture below, as well as the task name and the overall progress of the task. The table below describes all parts of the pert bar.

(1) The according row number of the pert bar

(2) The overall duration of the pert bar

(3) The number of the parent row

(4) The name of the task

(5) A visualization of the progress of the pert bar

(6) The numeric value of the pert bars progress

(7) The earliest start of the pert bar

(8) The latest start of the pert bar

(9) The earliest finish/end of the pert bar

(10) The latest finish/end of the pert bar

(11) The local free buffer time (in days) of a pert bar

(12) The global free buffer time (in days) of a pert bar.

The following table shows the properties of a pert bar:

Property	Description
`AdditionalSections`	A list that holds all additional section objects for this pert bar
`AdditionalSectionsFont`	The font that is used for the additional sections

`AdditionalSectionsStringFormat`	The string format that is used for the additional sections of the pert bar.
`BackgroundColor`	The background color for the pert bar.
`BackgroundSummaryColor`	The color used to visualize the progress of the pert bar
`BorderColor`	The color used for drawing the inner and outer lines of the pert bar
`BorderThickness`	The thickness of the lines in pixel.
`BottomSection`	A reference to the bottom section object.
`BottomSectionFont`	The font that is used for bottom section.
`BottomSectionStringFormat`	The string format that is used for the bottom section.
`Detailed`	If the TypeTimeSpan property has been set to true; the lower information panel (6-12) can be hidden if Detailed is set to false; if Detailed is true the lower information panel is visible.
`Height`	The overall height of the pert bar in pixel.
`MainSection`	A reference to the main section object of the pert bar
`MainSectionFont`	The font that is used for the main section.
`MainSectionStringFormat`	The string format that is used for the main section of the pert bar.
`ParentRow`	The row object for the pert bar.
`TextColor`	The color that is used when drawing the text of the pert bar.
`TimeSpanFont`	The font for the time span
`TimeSpanStringFormat`	The string format for the time span
`TopSection`	A reference to the top section object.
`TopSectionFont`	The font for the top section object.
`TopSectionStringFormat`	The string format for the top section of the pert bar.
`TypeTimeSpan`	If the TypeTimeSpan is set to false the pert bar shows the top-, main-, bottom- and additional sections. If a pert bar is in this mode, it is most customizable. If the TypeTimeSpan is set to true the pert bar shows time scheduling specific information, such as the earliest start and earliest finish etc.
`Width`	The overall with of the pert bar in pixel.
`X`	The x-position of the pert bar.
`Y`	The y-position of the pert bar.

Adding pert bars

There are different ways how to add a pert bar. Please keep in mind that pert and task bars are only different visual entities of the same task. This means whenever you have add a task bar to the gantt component by program logic or via user interface the corresponding pert bar will be created automatically.

When adding a pert bar, please keep in mind that there can only be one pert bar for each row. So if you add a new pert bar to a row that already has a pert bar, nothing will happen.

As it is the case for the GanttGraph the end user is able to create new bars by holding the left mouse button down and dragging a new object into the pert chart.

The code snippet below shows how to add a pert bar to the control:

- First we make a **`ChangeRightPart()`** statement that will force the component to turn into the pert view. If the gantt chart is already in the pert view mode, than ChangeRightPart() should not be called. After that, we create a new pert bar and specify the required parameters for the constructor. Finally, we do call the Refresh() method, to force the component to repaint itself.

(26)

```
…
ganttControl1.ChangeRightPart();

xmlDataSource1.Rows[1].PertBar = new PertBar(xmlDataSource1, // Datasource
         true, // TypeTimeSpan
         true, // Detailed
         new Font("Arial", 9), // TimeSpanFont
         new StringFormat(),   // String Format
         200, 175, // Width, Height
         System.Drawing.Color.Black, // Border Color
         1, // Border Thickness
         System.Drawing.Color.Beige, //Background Color
         System.Drawing.Color.Blue,  //Background Summ. Color
         System.Drawing.Color.Blue); //Text Color
```

```
ganttControl1.Refresh();
…
```

Accessing pert bars

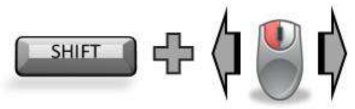
Pert bars can be accessed throughout the row object of the xmlDataSource as shown in the example below:

- In this example we access the pert bar of the first row and set the parameter `Detailed` to true.

(27)
```
…
xmlDataSource1.Rows[0].PertBar.Detailed = true;
…
```

Connecting pert bars

You can connect two pert bars by using the user interface interaction.



To do so, please move the mouse cursor over a pert bar, hold the [**Shift**] key down and press the left mouse button, move the mouse cursor to another pert bar and release the mouse button. After that, a new connection will be established.

For each pert bar you create - a task bar will be automatically added. Therefore you can simply connect the two referring task bars in order to connect two pert bars as it is shown in the chapter (**connection between bars**).

Print Preview

In the final stage of the workflow process it is often required to present and print out the created chart diagrams. A flexible print preview that is highly adaptable to the user's needs, allows the specification of lots of parameters (zoom, pagination, page title, background image).

Also a legend that is segmented into rows and columns can be attached to a diagram. Each cell can contain text or images. Furthermore the printing preview provides defining a header and footer –row, which is segmented into three areas.

Setting up the print preview

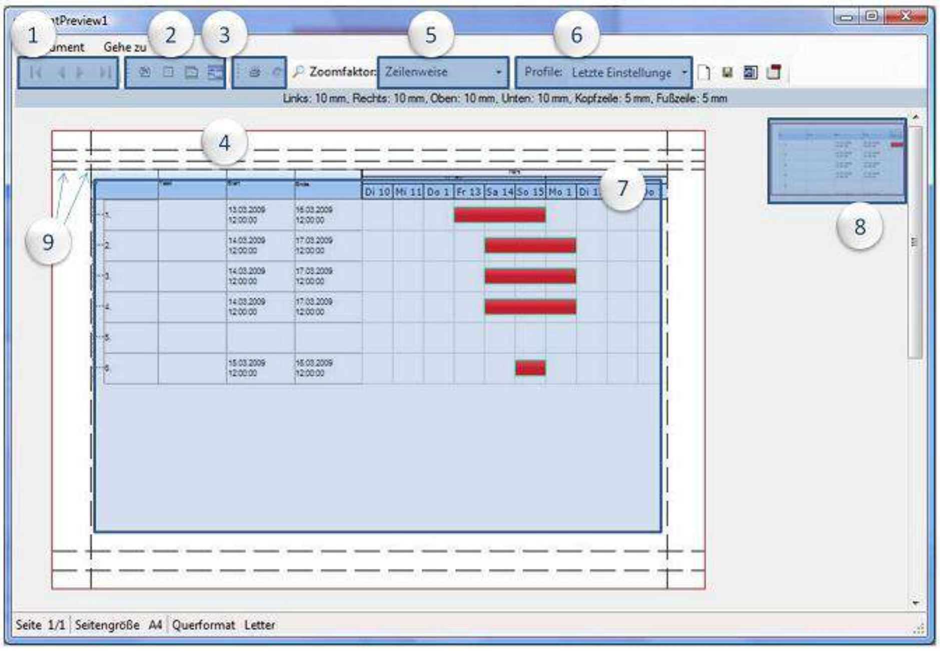
To set up the printing preview, please place a new PrintPreview component on you form. After that you have to connect the PrintPreview component with the datasource and link the treegrid, the gantt chart and the calendar to the print preview component. The following listing shows the necessary steps:

- In order to use the print preview, you have to link the xmlDataSource; the Graph; the Calendar and the TreeGrid to the print preview as shown here:

(28)
```
…
 printPreview1.DataSource = xmlDataSource1;
 printPreview1.GanttChart = ganttControl1.Graph;
 printPreview1.Calendar = ganttControl1.Graph.Calendar;
 printPreview1.TreeGrid = ganttControl1.TreeGrid;
…
```

- To invoke the printing preview just use the `ShowDialog()` statement.

(29)
```
…
printPreview1.ShowDialog();
…
```



Zoomfaktor: Zeilenweise

Profile: Letzte Einstellunge

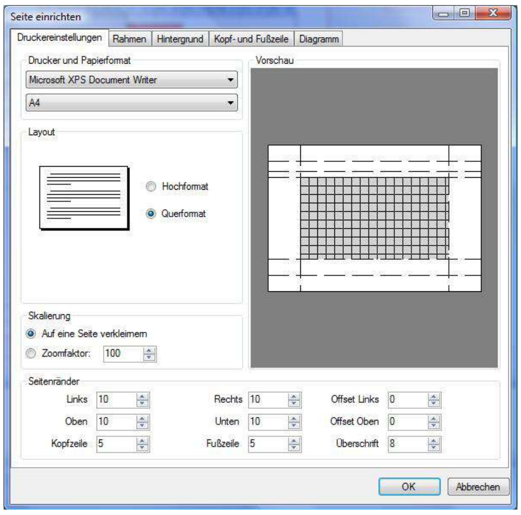
Links: 10 mm, Rechts: 10 mm, Oben: 10 mm, Unten: 10 mm, Kopfzeile: 5 mm, Fußzeile: 5 mm

1		13.03.2009 12:00:00	16.03.2009 12:00:00
2		14.03.2009 12:00:00	17.03.2009 12:00:00
3		14.03.2009 12:00:00	17.03.2009 12:00:00
4		14.03.2009 12:00:00	17.03.2009 12:00:00
5			
6		15.03.2009 12:00:00	16.03.2009 12:00:00

Seite 1/1 Seitengröße A4 Querformat Letter

The following picture shows an overview of the printing preview. The tool buttons in the area (#1) are used to navigate throughout the pages of the print preview. The tool buttons referred in the area (#2) are used to set up and format the content of the print preview. These are in detail from left to right: “Page caption”, “Background”, “Border”, “Page header and footer” and “Gantt chart options”. A click on each of those buttons shows a sub dialog, where the user can set up the layout options according to the buttons category. The tool button (#3) is used to print the content of the print preview. To show the page settings dialog you have to click on the tool button (#4). You can adjust a zoom factor for the preview by using the drop down combo box (#5). Please note that this zoom factor is only for the preview itself and does not have any effect on the printed results. The print preview also provides the usage of print profiles. Here you can store all settings in a print profile. The tool buttons in the area (#6) are used to manage print profiles, here you can create new profiles, delete and rename existing profiles or apply an existing print profile to the settings of the print preview. The content of the print preview is shown in the area (#7) and a thumbnail view of the containing pages in the area (#8). The page borders (#9) are displayed on the main view area of the print preview.

Page setup dialog



Drucker und Papierformat: Microsoft XPS Document Writer; A4

Layout: Hochformat [✓ Querformat]

Vorschau

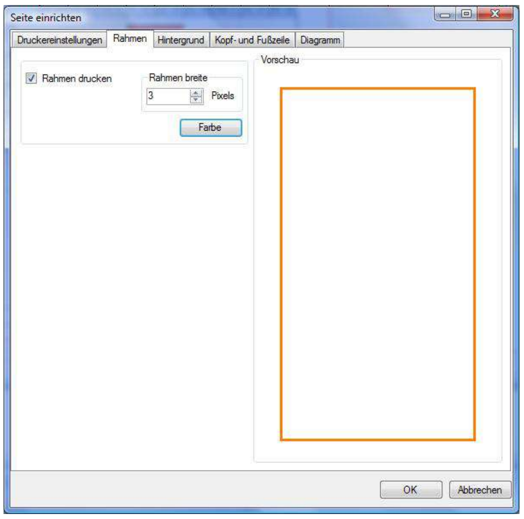
Skalierung: [✓ Auf eine Seite verkleimern] Zoomfaktor: 100

Seitenränder

Links	10	Rechts	10	Offset Links	0
Oben	10	Unten	10	Offset Oben	0
Kopfzeile	5	Fußzeile	5	Überschrift	8

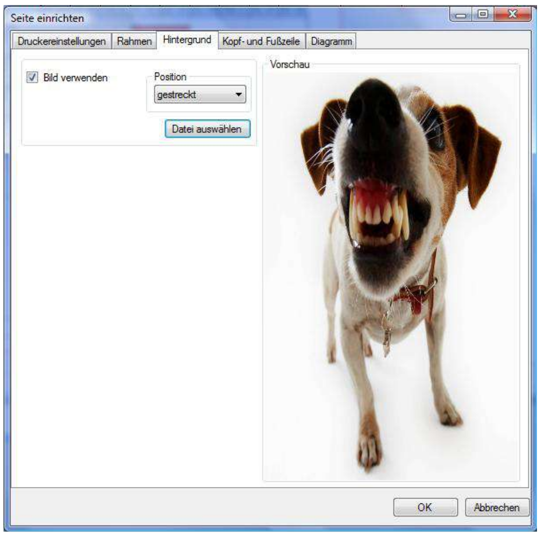
Here you can see the selected printer, the format and its orientation. Also you can specify whether the content should be stretched to fit one page (Fit to page) or if a zoom factor should be applied (Zoom factor) for printing. Please note, that this zoom factor affects the resulting print-out unlike the zoom factor affects only the print preview as it was shown above.

Also the user can specify the page borders at the bottom of the dialog. It is also possible to set up a global printing offset that shifts the print out according to the specified values to the right and/or to the bottom.



[✓] Rahmen drucken
Rahmen breite: 3 Pixels

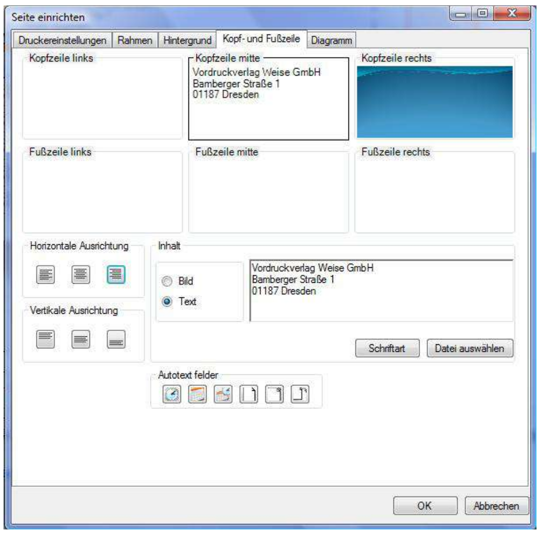
It is possible to print a border that surrounds the entire content. A color and the width of the border can be specified here.



[✓] Bild verwenden
Position: gestreckt

Here the user can define a background for the gantt chart or pert chart diagram. A background can be drawn using a solid fill background defined by its color.

Also it is possible to select an image file that is used as the background. If the user selects an image as the background he can select the referring draw mode (stretched, centered and tiled).



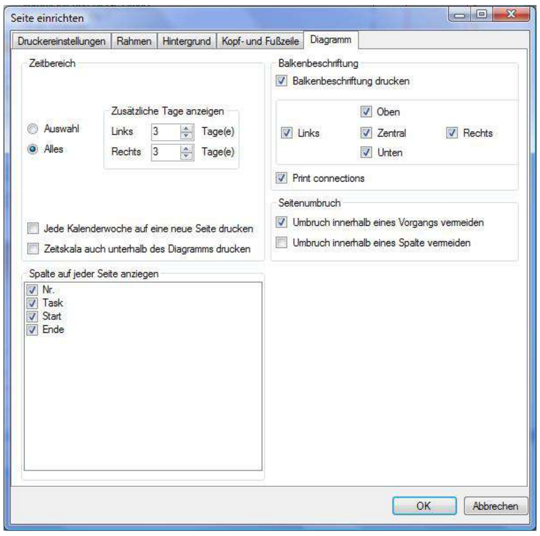
Kopfzeile mitte:
Vordruckverlag Weise GmbH
Bamberger Straße 1
01187 Dresden

Inhalt: Bild [✓ Text]

Vordruckverlag Weise GmbH
Bamberger Straße 1
01187 Dresden

At this point the page header and the page footer can be defined. Header and footer are divided into three sections. At the top of the dialog is a preview of the header and footer.

For each segment of the header or footer you can either type in some text or load an image that is displayed inside the segment. Also you can insert auto text fields into single segments. Auto text fields are responsible for displaying the current page number, the current time and so on.



Zeitbereich
- ○ Auswahl
- ● Alles

Zusätzliche Tage anzeigen
- Links: 3 Tage(e)
- Rechts: 3 Tage(e)

- ☐ Jede Kalenderwoche auf eine neue Seite drucken
- ☐ Zeitskala auch unterhalb des Diagramms drucken

Spalte auf jeder Seite anziegen
- ☑ Nr.
- ☑ Task
- ☑ Start
- ☑ Ende

Balkenbeschriftung
- ☑ Balkenbeschriftung drucken
- ☑ Oben
- ☑ Links
- ☑ Zentral
- ☑ Rechts
- ☑ Unten
- ☑ Print connections

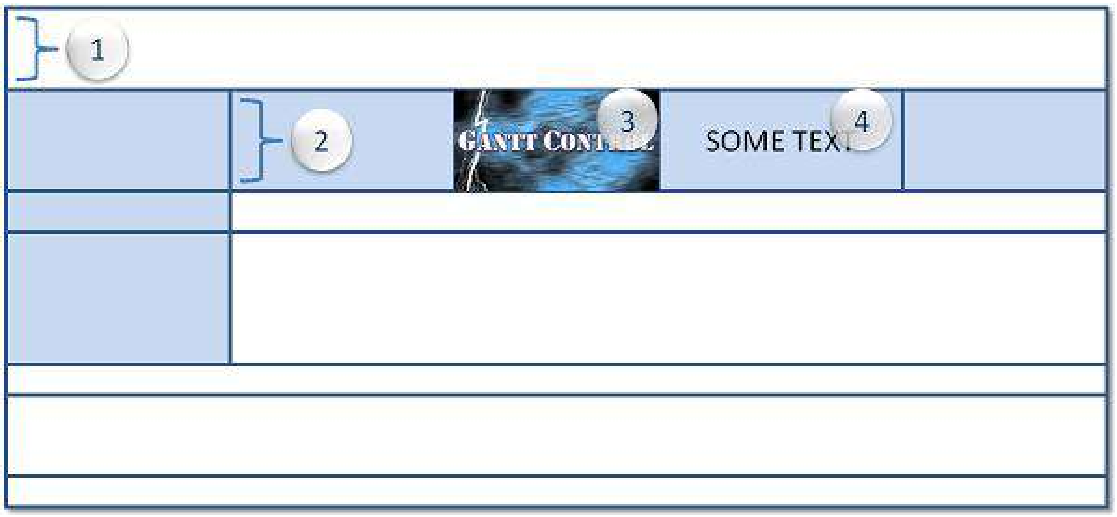
Seitenumbruch
- ☑ Umbruch innerhalb eines Vorgangs vermeiden
- ☐ Umbruch innerhalb eines Spalte vermeiden

The last section of the page setup dialog is only visible if you print a gantt chart. If so, you can specify the time range used for printing.

Also it’s possible to select columns that are printed on each page (if there exists more than one page).

Legend

To display any textual and graphical information you can create a legend. The legend will be displayed below the GanttGraph or the PertGraph. The architecture of a legend is based on a grid-like structure, including a set of rows (#1) and cells (#2) that are added to a row as shown in the picture below. The legend object is a list of rows, each row is a list of cells.



There are two different cell types that can be added to a row. The LegendImageCell (#3) can contain images and the LegendTextCell (#4) stores text information. Please note that a row does not need to have cells at all, also each row can own a different amount of legend cells.

The following table shows the most relevant operations and properties of the legend. The legend object can be access by the printpreview.

Property	Description
`Count`	Returns the total number of rows
`Size`	Returns the size in pixels of the legend
`BackgroundColor`	The background brush(!) for the legend
`BorderColor`	The border brush(!) for the legend
`BorderWidth`	The width in pixel of the border
`Render()`	This will finally render the legend
`Add(..)`	Adds a new legend row to the legend
`Clear()`	Clears all rows of the legend

`Insert(...)`	Inserts a new legend row for the specified position
`Remove(...)`	Removes a legend row

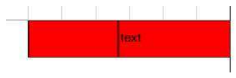
The rows objects provides the following properties and functions:

Property	Description
`Count`	Returns the total number of cells for this row
`Add(..)`	Adds a new cell to the row of the legend
`Width`	Returns the width of the row
`Height`	Returns the height of the row
`Clear()`	Clears all cells of this row
`Insert(...)`	Inserts a new cell to this row
`Remove(...)`	Removes a cell from the legend row

The legend cell objects provide the properties and functions shown below:

Property	Description	Legend cell type
`Alignment`	The Alignment of the cell within the row	
`Width`	The cells width	
`Font`	The cells font	LegendTextCell
`Text`	Text text that is displayed for the cell	LegendTextCell
`TextAlignment`	The alignment for the legend text cell	LegendTextCell
`Image`	The image for the legend image cell	LegendImageCell

The following code creates a very small legend.



- Please note, that we have added the namescape VVW.GanttControl.GanttPrintPreview.Legend first. After that, we create a new **`LegendBoard`** object and add a new row to it. Finally we add a **`LegendTextCell`** to the **`LegendRow`**.

(28)
```
using VVW.GanttControl.GanttPrintPreview.Legend;
…

LegendBoard legende = new LegendBoard();

legende.BorderColor = new SolidBrush(Color.Black);
legende.BackgroundColor = new SolidBrush(Color.Red);

LegendRow row1 = new LegendRow(legende);
LegendTextCell cell1 = new LegendTextCell(row1, "text");

printPreview1.Legend = legende;

…
```

Localization

Currently the Gantt Component is available for the both languages German and English. There are some parts of the gantt control component that are very language specific, such as the print preview. The print preview has a lot of user interface elements such as buttons, menus, labels etc. Therefore it might be necessary first to specify the language that should be used when showing the print preview.

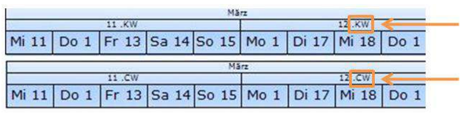
To localize the print preview you have to specify the `CurrentUICulture.` It is necessary to set up the CurrentUICulture before(!) the print preview component is created. The following code shows how to set up the CurrentUICulture scheme for English.

- After we have set up the new culture scheme, we have to recreate the print preview.

(28)

```
using System.Threading;
…

Thread.CurrentThread.CurrentUICulture = new System.Globalization.CultureInfo("en");
//Thread.CurrentThread.CurrentUICulture = new System.Globalization.CultureInfo("de");

if (printPreview1 != null)
     printPreview1.Dispose();

printPreview1 = new VVW.GanttControl.GanttPrintPreview.PrintPreview();

…
```

Another part, where the gantt control component needs to be localized is the language specific date format used in the time scale. The picture below shows some example where a hard coded string (“.KW”, “.CW”) needs to be reassigned.


März
11 .KW
12 .KW
Mi 11 | Do 1 | Fr 13 | Sa 14 | So 15 | Mo 1 | Di 17 | Mi 18 | Do 1
März
11 .CW
12 .CW
Mi 11 | Do 1 | Fr 13 | Sa 14 | So 15 | Mo 1 | Di 17 | Mi 18 | Do 1

In order to do so, you have to use the method **`SetDateStringFormat(..)`** of the Calendar. The first parameter specifies the string format and the second parameter specifies the time mode (day, week, month, etc.) for the string format.

- The code sample below changes the string for calendar weeks from “.CW” into “.KW”.

(28)

```
…
ganttControl1.Graph.Calendar.SetDateStringFormat("{0} .KW", Calendar.TimeModes.Week);
…
```

By default the **`DateStringFormat`** uses the following values:

StringFormat	TimeMode
`"HH"`	Hour
`"ddd d"`	Day
`"{0} .CW"`	Week
`"MMMM"`	Month
`"{0}. Quarter"`	Quarter
`"yyyy"`	Year
`"{0} - {1}"`	Decade

Events

The Gantt Control component defines several events that are accessible through your Development Environment. To access an event, please select the GanttControl component or the XMLDataSource. After that, open the “Properties-Dialog” [CTRL + W, P] and double click on the event to implement its event handler.

The following list shows all events of the GanttControl component:

GanttControl.BarAdded
The BarAdded event is raised as soon as the user adds a new bar. The event will be raised for task bars and progress bars.

GanttControl.BarDoubleClick
The BarDoubleClick event is raised when a user double clicks on a progress or a task bar.

GanttControl.BarMouseDown
The BarMouseDown event is raised when a user clicks a progress or a task bar.

GanttControl.ChartImageDoubleClick
The ChartImageDoubleClick event is raised when a user double clicks on a chart image.

GanttControl.ChartTextBarDoubleClick
The ChartTextBarDoubleClick event is raised when a user double clicks on a text bar.

The following list shows the events of the XMLDataSource.

RowActionStart
Before a row-operation is executed the RowActionStart event is raised. A row-operation is one of the following operations: (RowAppend, RowClear, RowInsert, RowDelete, IncreaseLevelOfRow, DecreaseLevelOfRow, BranchDelete, RowChildInsert, RowsGroup and RowsChangeParent)

RowActionEnd
After a row-operation is executed the RowActionEnd event is raised. A row-operation is one of the following operations: (RowAppend, RowClear, RowInsert, RowDelete, IncreaseLevelOfRow, DecreaseLevelOfRow, BranchDelete, RowChildInsert, RowsGroup and RowsChangeParent)

Update
Whenever changes are made to the GanttControl the Update event will be fired. The event holds the parameter **`EventArgs`** that can be used to retrieve the internal method that has fired the Update event.