

Gantt Control Developer Reference

DELPHI Language Guide

VERSION 2.044



Vordruckverlag Weise GmbH
Bamberger Str. 1
01187 Dresden
Germany
<http://www.gantt-komponente.de/>
<http://www.gantt-component.com/>

GANTT CONTROL

© Copyright 2007,2008 Vordruckverlag Weise GmbH – Germany

Table of content

Installation.....	4
Component Overview.....	5
Rapid Path Access.....	6
WGSanttGraph.....	8
<i>Preparing the GanttGraph.....</i>	8
<i>Time scales.....</i>	9
<i>Time mode.....</i>	9
<i>Width of a time unit.....</i>	9
<i>Row operations.....</i>	10
<i>Hierarchical structuring of rows.....</i>	11
<i>Adding columns to the tree.....</i>	11
<i>Accessing cells.....</i>	15
<i>Adding bars to the gantt table.....</i>	16
<i>Task bars.....</i>	18
<i>Progress bars.....</i>	22
<i>Milestone bars.....</i>	23
<i>Image bars.....</i>	24
<i>Text bars.....</i>	25
<i>Calendar.....</i>	26
<i>Critical Path.....</i>	28
<i>Connection between bars.....</i>	29
WGSDataSource.....	32
<i>Undo Redo Stack.....</i>	32
WGSPertGraph.....	34
<i>Pert bar (Type I – “No Time Span”).....</i>	35
<i>Pert bar (Type II – “Time Span”).....</i>	35
<i>Pert bar properties.....</i>	36
<i>Adding pert bars.....</i>	37
<i>Accessing pert bars.....</i>	37
<i>Deleting pert bars.....</i>	38
<i>Connecting pert bars.....</i>	38
<i>Formatting the pert graph.....</i>	39
KaTPrintPreview.....	40
<i>Setting up the Printing Preview.....</i>	40
<i>Printing Preview.....</i>	41
<i>Page Setup dialog.....</i>	42
<i>Print Options.....</i>	43
<i>Legend.....</i>	45
Localization.....	48
Events.....	51
Time format value table.....	56

This developer reference guide describes the software component gantt control VCL edition based upon the DELPHI language. For most of the examples provided in this reference you need to have the software component package installed to your Delphi IDE.

Installation

To install the gantt control to your DELPHI environment, do the following steps:

If you own the gantt control package not including the **full source code** (demo version) please to the following steps to install the package to your DELPHI IDE:

- Uninstall any previously installed gantt control components.

In DELPHI, go to *Components - Install Package* menu. From design-time packages list, please select WeiseGanttSuite and click *remove*. Go to *Tools-Environment Options - Library* page and remove the WeiseGanttSuite directory from your library path.

- Shut down DELPHI and all other applications.
- Download and extract the files from the gantt-control archive.
- Start DELPHI:

In DELPHI go to *Components - Install Package* menu - *Add ...*. Select the GanttSuiteVCL_D7, GanttSuiteVCL_D2005 or GanttSuiteVCL_D2006 according to your Delphi Version. Set the library path in DELPHI: Go to *Tools - Environment Options - Library* and add to the library path the path you have installed and extracted the GanttSuite.

If you own the gantt control package including the **full source code** please reproduce the following steps to install the package to your DELPHI IDE:

- Uninstall any previously installed gantt control components.

In DELPHI, go to *Components - Install Package* menu. From design-time packages list, please select WeiseGanttSuite and click *remove*. Go to *Tools-Environment Options - Library* page and remove the WeiseGanttSuite directory from your library path.

- Shut down DELPHI and all other applications.
- Download and extract the files from the gantt-control archive.
- Start DELPHI:

Open the GanttSuiteVCL_D7.dpk, GanttSuiteVCL_D2005.dpk or GanttSuiteVCL_D2007.dpk according to your Delphi version. Click **compile** and **install**. Go to *Tools - Environment Options - Library* and add to the library path the path you have installed and extracted the GanttSuite.

Component Overview

After correctly installing the gantt control package to Delphi you should see a new tab sheet in your component palette showing the following four components:



WSGanttGraph

The GanttGraph is an interactive user interface component that contains a tree grid and a gantt chart. The gantt chart diagram can contain bar objects that are used for scheduling. Each row of the gantt chart is linked to a row of the tree grid.



WSPertGraph

The PertGraph is an interactive user interface component that contains a tree grid and a pert chart. Pert charts can contain pert bars that symbolize specific entities and or tasks.



WGSDataSource

The DataSource component is a non visual component. Each WSGanttGraph and WSPertGraph component **has to** be connected to a WGSDataSource. The DataSource contains all data and provides functionalities and access methods to those data.



KATPrintPreview

The KaTPrintPreview is a non visual component that can be used to invoke a printing preview user interface. A KaTPrintPreview component **must** be linked to a DataSource.

The gantt control component package can be described best as an interactively front end control used to visualize and process information within a gantt or pert view. It does not include any database access control mechanism nor - any data base bound mode that automatically reads and writes the data from and to a database.

As there are some gantt control components that allow only displaying information, the gantt control does support displaying and editing data - interactively by the user interface or by application logic.

Rapid path access

In this chapter the basic steps will be introduced that are required to include the gantt chart in your delphi projects.

- After creating a new delphi project place a `WGSanttGraph`, a `WGSDataSource` and a `KATPrintPreview` component on your delphi form. If you have placed all components to the form you have to **connect** the `WGSanttGraph` and the `KATPrintPreview` to the `DataSource`.
- To do so, please select the `WGSanttGraph` component in the object inspector (**F11**) and assign `WGSDataSource1` for the `DataSource` property. Now repeat this step for the `KATPrintPreview` component too.
- In the object inspector select `alTop` for the `Align` property of the `GanttGraph`.

If you have accomplished those three steps you can compile and run your delphi application. You should now see a gantt chart diagram that does already have one row and the column `Nr`.

The programs user is now able to drag in a new bar or resize a bar with the mouse. Now we will extend the existing project and add some basic application logic for adding new columns, new rows and new bars.

- In your Delphi IDE press **F12** for the code editor - scroll to the top of your source file and go to the `uses` clause. Now it's necessary to add some more units to your delphi project. Please add the units `wgsTaskbar` and `wgsTreeSimpleText` cell to the `uses` clause - so that it looks like:

*(Note: If you receive an error while compiling the project you probably do not have set the proper source path. Go to **Tools - Environment Options - Library** and add to the library path the path you have installed and extracted the **GanttSuite**).*

```
(1) uses
    Windows, Messages, SysUtils, Variants, Classes,
    Graphics, Controls, Forms,
    Dialogs, StdCtrls,

    KATPrintPreview,
    wgsDataSource,
    wgsGanttEntities,
    wgsController,
    wgsGanttGraph,
    wgsTaskBar,
    wgsTreeSimpleTextCell;
```

- After the units are added we will add a new column when the form is created. To do so please open the object inspector (**F11**), select your form. The name is probably `Form1` if you do not have set another name and select the event tab page in the object inspector. Now double click on the forms `OnCreate` event and add the following two lines of Delphi code:

```
(2) procedure TForm1.FormCreate(Sender: TObject);
begin
    WGSDataSource1.ColumnAppend(TWGSTreeSimpleTextCellType.Create);
    WGSDataSource1.Tree.Header.Cells[1].Title := 'Name';
end;
```

- Now it is time to place a new button on your Delphi formular. Each time the user presses the button a new row including a new bar should be added to the gantt graph by application logic. Double click on the placed button and enter the following code:

```
(3) procedure TForm1.Button1Click(Sender: TObject);
var
    Bar : TWGSTaskBar;
    Row : Integer;
begin
    WSGanttGraph1.BeginUpdate;
    // Adding a new row
    WGSDataSource1.RowAppend;
    // The index of the last row
    Row := WSGanttGraph1.Tree.RowsCount-1;
    // Creating a new bar object
    Bar := TWGSTaskBar.Create(Now,Now+14,Row);
    // Add this Bar to the datasource
    WGSDataSource1.BarAdd(Bar,Row);
    WSGanttGraph1.EndUpdate;
end;
```

The next step will be to add some printing functionality. To do so we will add two more buttons. The first button is used to invoke the standard delphi printer setup dialog to select the printer and the printer format. The second button is used to call the printing preview.

- So place a new PrinterSetupDialog and a new button on the form. After clicking the button the PrinterSetupDialog will be executed.

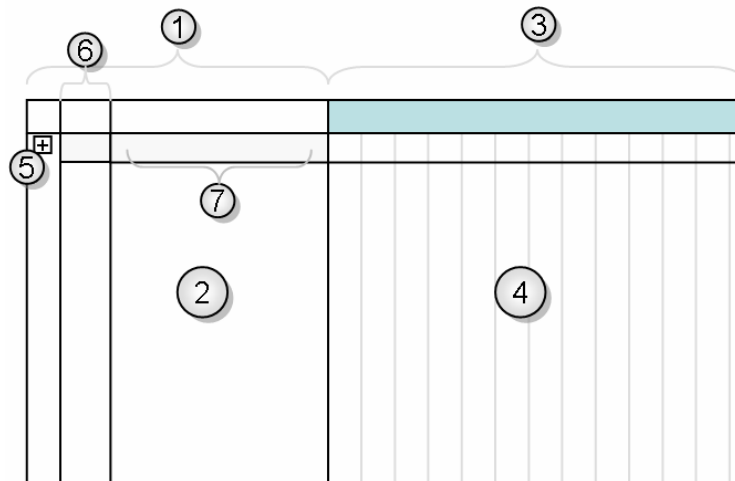
```
(4) procedure TForm1.Button2Click(Sender: TObject);
begin
    PrinterSetupDialog1.Execute;
end;
```

- To call the printing preview, enter the following Delphi code in the button3 OnClick event. Please check if you have already set the `DataSource` property of TKatPrintPreview component to WGSDataSource1. In the first two lines of code, we specify that the ganttchart and the treegrid of the ganttgraph should be printed. Finally `DoPrint` will call the PrintingPreview.

```
(5) procedure TForm1.Button3Click(Sender: TObject);
begin
    KaTPrintPreview1.PrintOptions.PrintGraph := true;
    KaTPrintPreview1.PrintOptions.PrintTable := true;
    KaTPrintPreview1.DoPrint(true,false)
end;
```

WGSanttGraph

The GanttGraph is an interactive user interface component that contains a tree grid and a gantt chart. The gantt chart diagram can contain bar objects that are used for scheduling. Each row of the gantt chart does have a row of the tree grid.



To identify the most elements of the gantt chart please see the following list:

- 4 The gantt chart area, also called (gantt)**table**. Elements of the gantt chart like bars, images, milestones are displayed within this area.
- 2 The hierarchical **tree grid**.
- 3 The **timescale(s)** is/are part of the gantt table.
- 1 The **header** of the tree grid.
- 5 A **tree node** that can be collapsed and expanded.
- 6 A tree **column**.
- 7 A tree **cell**.

Preparing the GanttGraph

In order to integrate the gantt graph into your software application and their specific requirements - it's necessary to adapt the gantt graph. Within this section the most relevant steps are described to set up a basic environment using the gantt graph.

When you have placed the ganttgraph component into your delphi form - the gantt graph does have by default only one time scale. You can add as many time scales as you want. In order to do so, add a new time scale by using the **InsertScale** method of the **document**.

- Assuming you have the ganttgraph named WGSanttGraph1 you can add a time scale using the following command: The first parameter specifies the index position where the time scale should be inserted. The second parameter specifies the time scale object that is inserted.

```
(6) With wgsganttgraph1.document do
    InsertScale(0, TWGSTimeScale.Create(tmMonth, WGSanttGraph1.document));
    ..
    Wgsganttgraph1.Repaint;
```

For adding and defining time scales its necessary to include the two units: **wgsTimeScale** and **wgsCalendar** to the uses clause of the formular.

Time scales

Each time scale does have its own time mode. The time mode describes the length of the time segments, the time scale will be segmented in. For example a time scale can visualize days or months if the time mode is set to `tmDay` or `tmMonth`. The time mode can have the following values of the set `TWGSTimeMode` defined in the unit `wgsCalendar`:

<code>tmHour</code>	Hour
<code>tmDay</code>	Day
<code>tmWeek</code>	Week
<code>tmMonth</code>	Month
<code>tmQuarter</code>	Quarter
<code>tmYear</code>	Year
<code>tmDecade</code>	Decade

For formatting a time scale you can specify the following properties:

Property	Description
<code>Height</code>	Height describes the height of the timescale in pixel.
<code>Color</code>	The background color of the time scale.
<code>FontColor</code>	The font color used within the time scale.
<code>LineColor</code>	Color that is used for segmenting single time units (e.g. days)
<code>FontHeight</code>	The font size used within the time scale.
<code>TimeFormat</code>	Time Format specifies the display format of the time unit.

Time mode

The calendar is the main module holding the functionality for changing the global time mode of the gantt control component and defining working and non working times. Whenever you change the time mode of the calendar - the time modes of the single time scales that are defined in the `ganttgraph.document` are readapted.

Example: If you create three time scales and set their time modes to (`tmWeek`, `tmMonth`, `tmYear`) and change the time mode of the calendar from `tmWeek` to `tmDay` - the time modes of the time scales will change to (`tmDay`, `tmWeek`, `tmMonth`). The time mode of the calendar describes the "smallest" time mode of all containing time scales.

To alter the time mode of the calendar you can use the `SetTimeMode` method.

- Assuming that the GanttGraph is named `WGSGanttGraph1` and your DataSource component is named `WGSDDataSource1` you can change the global time mode of the calendar using the `SetTimeMode` method as you can see in the example below: When using the example you have to insert the unit `wgsCalendar` to the unit clause of your form.

```
(7) ..
    WGSDDataSource1.Calendar.SetTimeMode(tmMonth);
    WGSGanttGraph1.Repaint;
..
```

Width of a time unit

As you have inserted some time scales and changed their time modes - you may have noticed that the width of a time unit (day, week ...) does have a predefined value. If you want to change the width of the time unit you can use the `ColumnWidth` property of the table.

- The `ColumnWidth` property of the Table specifies the width of a single time unit within the time scale that does have the "smallest" time mode. The time mode of the time scale that does have the smallest time mode always corresponds with the time mode of the calendar.

```
(8) ..
    WSGanttGraph1.Table.ColumnWidth := 20;
    WSGanttGraph1.Repaint;
..
```

Row operations

All data-sensitive operations (like adding a row) are performed by using the data source component. So the data source provides methods for adding deleting rows, bars and columns.

- For adding new rows use the **RowAppend** method of the datasource.

```
(9) ..
    WSGDataSource1.RowAppend;
    WSGanttGraph1.Repaint;
..
```

Whenever you plan to manipulate a lot of data, e.g. adding a lot of rows we recommend placing the operation between a **WSGanttGraph.BeginUpdate** and a **WSGanttGraph.EndUpdate** statement to enhance the performance of your software application.

- For deleting an existing row you can use the **RowDelete (Index: Integer)** function. Index specifies the row index, where the top most row has an index of 0.

```
(10) ..
    WSGDataSource1.RowDelete(1);
    WSGanttGraph1.Repaint;
..
```

Note: The gantt graph component does at least always contain one row at minimum. This means when you delete all rows - the top most row will not be deleted but cleared instead.

- If you want to delete only the content of a single row, this means deleting all cell values of this row and all bars linked to this row - you can use the **RowClear (Index: Integer)** function.

```
(11) ..
    WSGDataSource1.RowClear(1);
    WSGanttGraph1.Repaint;
..
```

- If you intend to insert a row at a given position you can use the **RowInsert (Index: Integer)** function.

```
(12) ..
    WSGDataSource1.RowInsert(1);
    WSGanttGraph1.Repaint;
..
```

Rows are logically separated into two parts - the row that is part of the tree grid (**TWGSTreeRow**) and the row that is part of the gantt table (**TWSGanttRow**).

For accessing single cells (and their values) as well as formatting the layout settings of a row you have to access the row as a **TWGSTreeRow** object. You may access a tree row by using the indexed row array of the tree (**WSGanttGraph.Tree.Row[n]**).

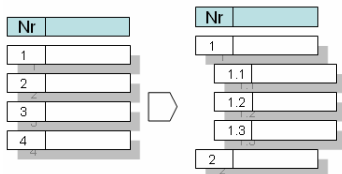
Rows of the gantt table may be accessed by using the gantt row (**WSGanttGraph.Table.Row[n]**) accessor of the table object.

Hierarchical structuring of rows

The tree of the gantt graph allows it to create a hierarchical structure. There are methods within the DataSource providing the developer to apply a hierarchical tree structure.

- You can group a range of rows and increase their nesting level by using the **RowsGroup(startIndex, endIndex:Integer)** method. The range is determined by the startIndex and the endIndex.

```
(13) ..
      WGSDataSource1.RowsGroup(0,2);
      WSGanttGraph1.Repaint;
      ..
```



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.

- Use the **RowChildInsert(Index:Integer)** to add and insert a new child row to an existing row for the specified index.

```
(14) ..
      WGSDataSource1.RowChildInsert(0);
      WSGanttGraph1.Repaint;
      ..
```

- For deleting all child rows of a row you can use the **BranchDelete(Index:Integer)** function.

```
(15) ..
      WGSDataSource1.BranchDelete(0);
      WSGanttGraph1.Repaint;
      ..
```

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

```
(16) ..
      WGSDataSource1.RowsChangeParent(1,2,0);
      WSGanttGraph1.Repaint;
      ..
```

Basically there are two different modes how the hierarchical structure is visualized in the "No"-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" ...) by setting the **WGSanttGraph.Tree.ContinuousNumbering** to true.

Whenever you add or delete (child)rows - the values of the "No"-column are automatically updated.

Adding columns to the tree

By default the tree of the gantt graph only includes the "No" column that shows a unique number value for each row. When adapting the gantt control to your specific requirements its necessary to add columns to the tree.

There are two 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. For example the "No"-column is a predefined column.

The following table shows all predefined columns, the unit they are declared and a brief description.

Predefined Column	Unit	Description
TWGSTreeAutoCellNumber	wgsTreeAutoCellNumber	The number column displays the current number of the row. There are two different display formats defined by the <code>GSGanttGraph.Tree.ContinuousNumbering</code> flag.
TWGSTreeAutoCellStartDate	wgsTreeAutoCellStartDate	The column "TreeAutoCellStartDate" displays the earliest start date of all bars for each row.
TWGSTreeAutoCellEndDate	wgsTreeAutoCellEndDate	The column "TreeAutoCellEndDate" displays the latest end date of all bars for each row.
TWGSTreeAutoCellTaskName	wgsTreeAutoCellTaskName	The column "TreeAutoCellTaskName" displays the task name of pert bars.

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 column, their unit and a brief description.

User defined Column	Unit	Description (Cell editor UI Component)
TWGSTreeSimpleTextCell	wgsTreeSimpleTextCell	The TreeSimpleTextCell column is used to edit or display a single line of text strings. (<i>TEdit</i>)
TWGSTreeMultilineTextCell	wgsTreeMultilineTextCell	The TreeMultilineTextCell column is used to edit or display multi line text. (<i>TMemo</i>)
TWGSTreeDateTimeCell	wgsTreeDateTimeCell	The TreeDateTimeTextCell column can be used for editing and displaying date and time values.
TWGSTreeSpinCell	wgsTreeSpinCell	The TreeSpinCell column is used to edit and display numeric values within a spin edit field. (<i>TSpinEdit</i>)
TWGSTreeCurrencyCell	wgsTreeCurrencyCell	The TreeCurrencyCell column is used to edit and display currency values. (<i>TMaskEdit</i>)
TWGSTreeComboCell	wgsTreeComboCell	The TreeComboCell column provides an editable combobox. (<i>TComboBox</i>)
TWGSTreeImageComboCell	wgsTreeImageComboCell	The TreeImageComboCell column provides an editable combobox including an additional image.
TWGSTreeButtonEditCell	wgsTreeButtonEditCell	The TreeButtonEditCell column is a composition of a TEdit field for editing and displaying simple text strings and an addition button.

Keep in mind that it's necessary to add the columns unit name to the unit clause of your formular when accessing column properties or add, delete columns.

For each column a cell **type** is declared within the columns unit. The cell type defines specific properties for the column based on the data type.

Example: For the column TWGSTreeSpinCell the cell type class **TWGSTreeSpinCellType** is implemented in the unit `wgsTreeSpinCell`. The class **TWGSTreeSpinCellType** manages data sensitive properties like *Min* and *Max* for the SpinEdit editor of the column.

Also unspecific cell type properties that are applicable to all types of cells can be accessed throughout the corresponding cell-type class (for example the ReadOnly-flag).

The following table show a list of all columns and their cell-type class:

User defined Column	Cell type class
TreeAutoCellNumber	TWGSTreeAutoCellNumber
TreeAutoCellStartDate	TWGSTreeAutoCellStartDate
TreeAutoCellEndDate	TWGSTreeAutoCellEndDate
TreeAutoCellTaskName	TWGSTreeAutoCellTaskNameType
TWGSTreeSimpleTextCell	TWGSTreeSimpleTextCellType
TWGSTreeMultilineTextCell	TWGSTreeMultilineTextCellType
TWGSTreeDateTimeCell	TWGSTreeDateTimeCellType
TWGSTreeSpinCell	TWGSTreeSpinCellType
TWGSTreeCurrencyCell	TWGSTreeCurrencyCellType
TWGSTreeComboCell	TWGSTreeComboCellType
TWGSTreeImageComboCell	TWGSTreeImageComboCellType
TWGSTreeButtonEditCell	TWGSTreeButtonEditCellType

When adding a column to the tree you have to create the cell type class for the column you want to add and use the **ColumnAppend** method of the datasource.

- The code snippet below shows how to add a simple text cell. As mentioned before its necessary to add the unit `wgsTreeSimpleTextCell`, create the `TWGSTreeSimpleTextCellType` class and call the **ColumnAppend** procedure. After the column has been created we will give a name to the new column using the **Title** property of the **cells**.

```
(17) uses ...,
      wgsTreeSimpleTextCell;

..
// Adding a new column
var
  SimpleTextCellType : TWGSTreeSimpleTextCellType;
begin
  SimpleTextCellType := TWGSTreeSimpleTextCellType.Create;
  WGSDataSource1.ColumnAppend(SimpleTextCellType);
  // Now we have to give a name to the new column
  with WGSDataSource1.Tree.Header do
    Cells[WGSDataSource1.Tree.Header.CellsCount-1].Title := 'New Column';
  WSGanttGraph1.Repaint;
end;
```

Based on this proceeding all other types of columns can be added to the tree. For the column `TWGSTreeComboCell` we will show how to add items to the drop down list of the combo box. Also for the column `TWGSTreeImageComboBox` we will show code examples for setting up an image combo box column.

Completing this chapter we will show how to specify a `ButtonEditCell` column this way that further program logic can be bound to the buttons `onClick` Event.

- After adding a `TreeComboCell` we will add some items to fill the combo box. The `Items` property that is of the type `TStrings` is defined within the type class `TWGSTreeComboCellType`.

```
(18) uses ...,
      wgsTreeComboCell;

..
// Adding a new column
var
  ComboCellType : TWGSTreeComboCellType;
```

```

begin
  ComboCellType := TWGSTreeComboCellType.Create();
  // Now we add some items to the combo box
  ComboCellType.Items.Add('item1');
  ComboCellType.Items.Add('item2');
  WGSDataSource1.ColumnAppend(ComboCellType);

  // Now we have to give a name to the new column
  with WGSDataSource1.Tree.Header do
    Cells[WGSDataSource1.Tree.Header.CellsCount-1].Title := 'New Column';
  WSGanttGraph1.Repaint;
end;

```

When using an ImageComboBox column, it is recommended to store all images you want to display into a TImageList component. In order to reproduce the following code – please place a new TImageList component on your Delphi formular and add some bitmaps to it.

- Assuming that you have filled an ImageList that is named “ImageList1” with 2 bitmaps the following code will add an imagecombobox column including two new items.

```

(19) uses ...,
      wgsTreeImageComboCell;

..
// Adding a new column
var
  ImageComboType : TWGSTreeImageComboCellType;
begin
  ImageComboType := TWGSTreeImageComboCellType.Create();
  // Now we add some items to the combo box
  ImageComboType.Images := ImageList1;
  ImageComboType.Items.Add('item1');
  ImageComboType.Items.Add('item2');
  WGSDataSource1.ColumnAppend(ImageComboType);

  // Now we have to give a name to the new column
  with WGSDataSource1.Tree.Header do
    Cells[WGSDataSource1.Tree.Header.CellsCount-1].Title := 'New Column';
  WSGanttGraph1.Repaint;
end;

```

After adding and configuring combo- and image comboboxes columns we will add a tree button edit cell and bind its OnClick Event to a defined procedure.

- In order to do so - we create a new procedure called “ButtonClick”.

```

(20) uses ...,
      dialogs,
      wgsTreeButtonEditCell;

..
Type
  TForm1 = class(TForm)
  ..
  private
    procedure ButtonClick(Target: TWGSTreeButtonEditCell);
  end;
  ..

procedure TForm1.ButtonClick(Target : TWGSTreeButtonEditCell);
begin
  ShowMessage('The Button was clicked');
end;
..

// Adding a new column
var

```

```

ButtonType : TWGSTreeButtonEditCellType;
begin
  ButtonType := TWGSTreeButtonEditCellType.Create();
  ButtonType.OnClick := ButtonClick;
  WGSDataSource1.ColumnAppend(ButtonType);

  // Now we have to give a name to the new column

  with WGSDataSource1.Tree.Header do
    Cells[WGSDataSource1.Tree.Header.CellsCount-1].Title := 'New Column';
  WSGanttGraph1.RePaint;
end;

```

The OnClick event of the TreeButtonEdit cell provides the clicked cell as the parameter: Target. In some cases it might be helpful if you can determine the row index and the column index of the cell. For example if you intend to write some value to the clicked cell.

- The row index of the target cell can be determined using the following statement:

```
(21) RowIndex := TWGSTreeRow(Target.Parent).RowIndex;
```

- The column index of the target cell can be determined using the following statement:

```
(22) ColIndex := Target.ColumnIndex;
```

To insert or delete a column the data source component defines the two functions **ColumnDelete(Index:Integer)** and **ColumnAdd(Index:Integer; DataType:TWGSTreeCellType)**.

Accessing cells


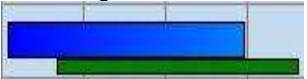


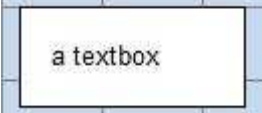
For accessing a single cell of the tree grid - the row array of the tree grid provides access to its cells. Note that when setting a value to a cell the columns data type must fit the cells value. When accessing a cell you may have to typecast the cell as the cell array returns the TWGSTreeCell object that is the base class for all derived cell classes. See the code example on how to access a TWGSTreeSimpleTextCell.

- For this example we want to access the second cell on the second row. To make the code more robust we check if there is a second column and a second row. After that we will test if the cell is a TWGSTreeSimpleTextCell and – if so – we will set a new text value.

```
(23) var
  Row, Col : integer;
begin
  Row := 1;
  Col := 1;
  if (WGSanttGraph1.Tree.RowsCount>Row) and
    (WGSanttGraph1.Tree.Header.CellsCount>Col) then
    begin
      if (WGSanttGraph1.Tree.rows[Row].Cells[Col]) is
        TWGSTreeSimpleTextCell then
        begin
          TWGSTreeSimpleTextCell(WGSanttGraph1.Tree.
            rows[Row].Cells[Col]).Text := 'hello';
        end;
      WGSanttGraph1.RePaint;
    end;
end;
```

Adding bars to the gantt table

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

Bar / Object	Description
	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 if form of a single connection. Taskbars are basically defined and determined by their StartDate and EndDate as well as the row in which it is displayed.
	A progressbar is commonly used to visualize the effective progress of a task. Therefore it is displayed under a task bar in the same row as the taskbar. 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.
	Milestones represent special dates with a unique character, for example a deadline where a project has to reach a certain status. A milestone will be represented as a small glyph that visualizes a specified time/date. Milestones do have a fixed size and are not resizable within the gantt graph.
	When adding an image to the gantt chart the TWGSImageBar type can be used.
	Any textual information can be visualized using the TWGSTextBar type.

The end user is able to create new bars by holding the left mouse button down and dragging a new object into a row of the gantt graph. By default all new bars that will be inserted this way (by the user) are task bars.

You can use the `setCreatedType(t:TWGSGanttBarClass)` of the gantt graph component to specify which kind of bar will be created.

- The following statement will change the bar kind to milestone bars.

```
(24) ...
      WGS GanttGraph1.SetCreatedType (TWGSMileStoneBar) ;
      ...
```

The following table summarize all possible bar types and their corresponding TWGSGanttBarClass that is used as the parameter for the `SetCreatedType` method and the unit the bar is defined.

Bar type	TWGSGanttBarClass	Unit
Taskbar	TWGSTaskbar	wgsTaskBar
Progressbar	TWGSProgressbar	wgsProgressBar
Milestonebar	TWGSMilestonebar	wgsMileStoneBar
Imagebar	TWGSImagebar	wgsImageBar
Textbar	TWGSTextbar	wgsTextBar

The data source component provides different methods processing bars. For adding new bars, use the `BarAdd (Bar:TWGSGanttBar; Row:Integer)` method. To delete an existing bar – use the `BarDelete (Bar:TWGSGanttBar)` function. If you intend to modify the time range or the row index of an existing bar use the `BarMove (Bar:TWGSGanttBar; StartDate:TDateTime; Row:integer)` or the `BarResize (Bar:TWGSGanttBar; DeltaX, DeltaY:Integer; RightSide:Boolean)` functions.

- When using the BarAdd method you need to assign the bar that should be added and the row where the bar should be inserted in. The code snippet below shows how to create and add a new task bar with a length of four days, starting with the current date. To reproduce this example, please add the unit `wgsTaskBar` to the use clause of your form. Also ensure that your GanttGraph contains at least two rows as the bar will be inserted to the second row (`RowIndex=1`).

The constructor of the `TWGSTaskBar` class requires the parameters `StartDate`, `EndDate` and the `Row Index`: `TWGSTaskBar.Create(StartDate, EndDate : TDateTime; Row:Integer)`.

```
(25) uses ...,
      wgsTaskBar;

      // Adding a new task bar
var
  Row:Integer;
  TaskBar : TWGSTaskBar;
begin
  Row:=1;
  TaskBar := TWGSTaskBar.Create(Trunc(now), Trunc(Now)+4, Row);
  WSGanttGraph1.BeginUpdate;
  WGSDataSource1.BarAdd(TaskBar, Row);
  WSGanttGraph1.EndUpdate;
end;
```

Depending on the row height and the height of a bar – some objects like image bars and text bars can cover more than one row. In this case the topmost row of all rows that the bar enfolds - forms the row index.

As shown in the example above the task bar is scheduled to the start of days. This means the taskbar starts and ends always 00:00. Of course you can also schedule bars using hours and minutes. However it may be advisable to change the `TimeMode` to `tmHour`, as it is easier for the user to recognize single hours on the time scale.

- The following code changes the overall time mode of the calendar to `tmHour`. Afterwards it creates a bar and adjusts the start date of the ganttgraph.

```
(26) uses ...,
      wgsTaskBar;
      // Adding a new task bar in "hourly mode"
var
  TaskBar : TWGSTaskBar;
  startDate,
  endDate : TDateTime;
begin
  WSGanttGraph1.DataSource.Calendar.SetTimeMode(tmHour);

  startDate := StrToDateTime('24.01.2008 10:13:00');
  endDate   := StrToDateTime('24.01.2008 23:45:00');

  TaskBar := TWGSTaskBar.Create(startDate, endDate, 0);
  WSGanttGraph1.DataSource.BarAdd( TaskBar, 0);

  WSGanttGraph1.document.setStartDate(Trunc(startDate));
  WSGanttGraph1.Repaint;
end;
```

For accessing bars there are different approaches. Basically all bars are accessible by its corresponding row.

- Each row (TWGSGanttRow) of the table does provide the bars array. The following code example shows, how to access a taskbar using the bars array of the row. First we verify that there is at least one bar on the first row. After this, we check whether the first bar of the first row is a taskbar – if so we typecast this bar as a TWGSTaskBar.

```
(27) uses ...,
      wgsTaskBar;

begin
  if WGSganttGraph1.Table.Rows[0].BarCount > 0 then
    if (WGSganttGraph1.Table.Rows[0].Bars[0] is TWGSTaskBar) then
      with TWGSTaskBar(WGSganttGraph1.Table.Rows[0].Bars[0]) do
        begin
          // Access the task bar here ...
        end;
      end;
    end;
  end;
```

Each bar does have the **IndexInRow** property that specifies the index of the bar within the bar-array of the row.

There are some further helpful constructions of the **TWGSGanttRow** class for accessing single bars. See the following table:

Function	Description
IndexOf (Bar:TWGSGanttBar):integer	IndexOf returns the index of the specified bar object within the bar array of the row. If the bar was not found within this row -1 is returned.
CountBarsOfKind(kind:TClass):integer	Returns the number of bars that are of the specified class. For example: Use CountBarsOfKind(TWGSTaskBar) to return the total number of taskbars that are on this row.
getBarOfKind(kind:TClass; i:integer):TWGSGanttBar	Returns the Gantt Bar of the specified class and the specified index position. Further typecasting of the returned TWGSGanttBar may be necessary.
BarCount:Integer	Returns the total number of bars within this row.
Bars[]:TWGSGanttBar	Accessible array of gantt bars that are within this row.

Taskbars

As shown above the GanttGraph provides five bar types. The most important – as this is the basic bar of gantt diagrams - is the taskbar. This chapter concentrates mainly on the properties a task bar contains and possibilities to change the layout of a taskbar. If you want to know on how to add a taskbar or how to access a taskbar – please have a look at the prior chapter (-> [adding bars to the gantt table](#)).

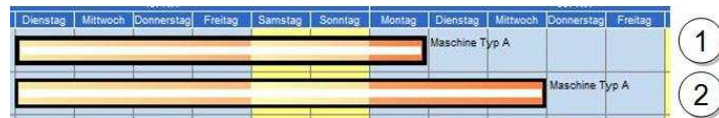
A taskbar is defined by its start and end date. Use the `WGSTaskBar.StartDate:TDateTime` or `WGSTaskBar.EndDate:TDateTime` property to assign a new value or read out the current value. If you want to assign a new start or end date you can also use the `WGSTaskBar.SetStartDate(d:TDateTime)` or the `WGSTaskBar.SetEndDate(d:TDateTime)` method. Please note that when assigning a new time range for taskbars the `EndDate` must always been larger than the `StartDate`. For example: If you want to change the duration of a task bar to one day the `EndDate` should be `EndDate := StartDate + 1;`

The duration of a taskbar can be obtained by readout the property `WGSTaskBar.Duration:TDateTime`.

In most cases you 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 directly can read out the time, the taskbar fits to working times using the

`WGSTaskBar.EffectiveDuration:TDatetime` property. Unlike the `Duration` property, that is read-only, you can also assign new values to 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 to allocation of different machines. Every time you allocate a new machine, the machine needs some reserved time to be built up and 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 2 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: `WGSTaskBar.bufferLeftTime:TDatetime` and `WGSTaskBar.bufferRightTime:TDatetime`. Note: Please ensure that the buffer times assigned to a bar are not larger than the duration of the bar itself, as for this case the GanttGraph is unable to display the correct duration of the buffer times.

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

Property	Description
<code>StartDate : TDatetime</code>	The start date of a taskbar.
<code>EndDate : TDatetime;</code>	The end date of a taskbar.
<code>Duration : TDatetime;</code>	(read only) The overall duration of the taskbar.
<code>EffectiveDuration:TDatetime;</code>	The duration of working time, based upon the calendar settings, of the taskbar.
<code>BufferLeftTime:TDatetime</code>	The buffer time for the start of the taskbar.
<code>BufferRightTime:TDatetime</code>	The buffer time for the end of the taskbar.

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



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

In most cases it is useful if you can link further textual information to a taskbar. For this purpose the `TWGSTaskBar` objects provides the `textHalo` object. You can assign 5 text

strings to a taskbar differing in their position and a hint string. See the following table gives a summary about all properties.

Property	Description								
<code>TextHalo.TextLeft:String</code>	A text - left of the taskbar								
<code>TextHalo.TextTop:String</code>	A text displayed above the taskbar								
<code>TextHalo.TextRight:String</code>	A text – right of the taskbar.								
<code>TextHalo.TextBottom:String</code>	A text displayed below the taskbar								
<code>TextHalo.TextCenter:String</code>	A text displayed inside the taskbar.								
<code>TextHalo.HorzAlignmentCenter:Integer</code>	The horizontal alignment of the bar text that is displayed inside the bar (TextCenter).								
	<table> <tr> <th>Value</th><th>Alignment</th></tr> <tr> <td>0</td><td>Left</td></tr> <tr> <td>1</td><td>Right</td></tr> <tr> <td>2</td><td>Center</td></tr> </table>	Value	Alignment	0	Left	1	Right	2	Center
Value	Alignment								
0	Left								
1	Right								
2	Center								
<code>Hint:String</code>	A hint for the taskbar								

If you want to adjust the layout of the text of a taskbar you have to use the `TWGSTaskBar.TextHalo.TextSettings.Font:TFont` object that provides a lot of useful parameters for adjusting the font of the text.

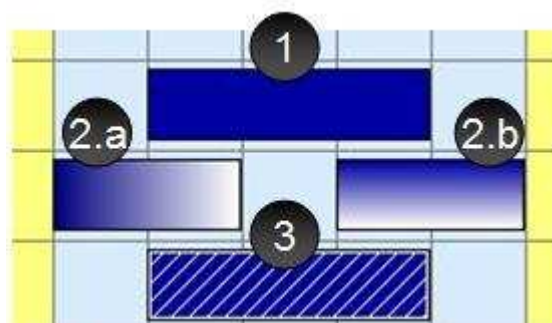
Note: If you want to access the properties of the `TextSettings` array, it is necessary to add the unit `wgsTextualHalo` of the Gantt Component Suite to your use clause.

In the rest of this chapter we will focus more on how to adjust the visual appearance of taskbars. All settings that affect the layout of a task bar are concentrated within the `TWGSTaskBar.Settings` and the `TWGSTaskBar.Settings.visualGoodie` objects. If you want to access some properties of those to object it is necessary to add the both units: `wgsTaskBarSettings` and `wgsVisualGoodie` to your use clause.

In the following table, all properties of the `TWGSTaskBar.Settings` object are described:

Property	Description
<code>Settings.BorderColor:TColor</code>	Each taskbar is surrounded by its border. Here you can define the color of the border.
<code>Settings.BorderThickness:Integer</code>	Specifies the width of the border in pixel.
<code>Settings.Height:Integer</code>	Here you can specify for each taskbar its height in pixel. Note: The taskbar can only be as high as its row is.
<code>Settings.ProgressColor:TColor</code>	The color that is used when displaying the internal bar progress.

There are three different styles a task bar can be drawn. These are a solid fill style (1) (`dsDrawSolid`), a gradient fill style (2) (`dsDrawGradient`) and a draw style using a pattern (3) (`dsDrawPattern`). For the gradient draw style you can also specify whether the gradient fill should be painted horizontal (2.a) or not (2.b).



According to the draw style you have selected for the taskbar, there are different properties that are used when drawing the bar.

<i>DrawStyle:= dsDrawSolid (1)</i>	
Property	Description
VisualGoodie.SFColor:TColor	Specifies the (solid fill) color of the taskbar.

<i>DrawStyle:= dsDrawGradient (2)</i>	
Property	Description
VisualGoodie.GFStartColor:TColor	Specifies the start color of the gradient fill
VisualGoodie.GFEndColor:TColor	Specifies the end color of the gradient fill
VisualGoodie.GFHorizontal:Boolean	Specifies the direction of the gradient fill.

<i>DrawStyle:= dsDrawPattern (3)</i>	
Property	Description
VisualGoodie.SFColor:TColor	Specifies the background color of the pattern
VisualGoodie.patternColor:TColor	Specifies the foreground color of the pattern
VisualGoodie.brushStyle:TBrushStyle	Specifies the brushstyle of the pattern. The object TBrushStyle is declared in the Delphi unit <i>Graphics</i> and can have one of the following values: <i>bsSolid</i> , <i>bsClear</i> , <i>bsHorizontal</i> , <i>bsVertical</i> , <i>bsFDiagonal</i> , <i>bsBDiagonal</i> , <i>bsCross</i> , <i>bsDiagCross</i> .

When the (end)user adds a new taskbar to ganttgraph or a taskbar is added by program logic its layout is defined by default settings. If you want to override those settings you can use the following code example shown below. Here we use the event `OnAfterBarAdd` of the `DataSource`.

- After adding a new bar the `OnAfterBarAdd` event of the `DataSource` will be raised. First we test here, if the added bar is a taskbar – if so we will change some layout parameters of the taskbar. To reproduce this example, please add the units `wgsGanttBar`, `wgsTaskBar`, `wgsTaskBarSettings` and `wgsVisualGoodie` to the use claus of your Delphi form.

To create the `OnAfterBarAdd` event, select the `DataSource` component on your Delphi formular and select “events” on the object inspector and double click the `OnAfterBarAdd` event.

```
(28) uses ...,
      wgsGanttBar,
      wgsTaskBar,
      wgsTaskBarSettings,
      wgsVisualGoodie;

...
procedure TForm1.WGSDatasource1AfterBarAdd(Bar: TWGSGanttBar);
begin
  if (Bar is TWGSTaskBar) then
  begin
    TWGSTaskBar(Bar).Settings.Height := 33;
    TWGSTaskBar(Bar).Settings.visualGoodie.drawStyle := dsDrawGradient;
    TWGSTaskBar(Bar).Settings.visualGoodie.GFStartColor := clYellow;
    TWGSTaskBar(Bar).Settings.visualGoodie.GFEndColor := clRed;
    TWGSTaskBar(Bar).textHalo.TextRight := 'taskbar';
  end;
end;
```

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



If you want to assign a new `StartGlyph` or `EndGlyph` you first have to create a new `TWGSPictureResource` object and assign it to the `StartGlyph` / `EndGlyph` of the taskbar. After that you have to create a `TGraphic` object (e.g. `TBitmap`) and assign it to the `StartGlyph.Picture` / `EndGlyph.Picture` object. Now you are ready to assign an image to the picture. This may sound a little bit difficult, therefore the previously code example will be extended so that an image will be assigned to the `StartGlyph` of the task bar.

- As we have to create a new `TWGSPictureResource` please add the unit `wgsResources` to your `uses` clause of the delphi formular. In the code snippet below, we directly load an image from a file and assign it to the `startGlyph`.

```
(29) uses ...,
      wgsGanttBar,
      wgsTaskBar,
      wgsTaskBarSettings,
      wgsVisualGoodie,
      wgsResources;

...
procedure TForm1.WGSDatasourceAfterBarAdd(Bar: TWGSGanttBar);
begin
  if (Bar is TWGSTaskBar) then
    begin
      TWGSTaskBar(Bar).Settings.Height := 40;
      TWGSTaskBar(Bar).Settings.visualGoodie.drawStyle := dsDrawGradient;
      TWGSTaskBar(Bar).Settings.visualGoodie.GFStartColor := clYellow;
      TWGSTaskBar(Bar).Settings.visualGoodie.GFEndColor := clRed;
      // Here we add a new startglyph to the taskbar
      TWGSTaskBar(Bar).Settings.visualGoodie.startGlyph := TWGSPictureResource.Create('');
      TWGSTaskBar(Bar).Settings.visualGoodie.startGlyph.Picture := TBitmap.Create;
      TWGSTaskBar(Bar).Settings.visualGoodie.startGlyph.Picture.LoadFromFile('D:\xmpl.bmp');
      TWGSTaskBar(Bar).TextHalo.TextRight := 'taskbar';
    end;
end;
```

Note: The Size of the Glyph is determined by the height of the bar. So it is recommended to use glyphs that's content can be recognised even if the glyph is very small, or to scale up the height of the row and the height of the bar in order to produce acceptable results.

Progressbars

Progressbars can be used to visualize the actual progress a task has made. For a better comparison they are directly drawn under the taskbar, so they may be applicable for any kinds 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 progressbar you are not restricted in any way – you can create as many progressbars for each row as you want. The amount of progressbars does not depend on the amount of taskbars, nor is a progressbar in anyway linked to a taskbar. If you want to know on how to add a progressbar or how to access a progressbar – please have a look at the prior chapter (-> [adding bars to the gantt table](#)).

As well as the taskbar the progressbar is defined by its `StartDate` and its `EndDate`. Use the `StartDate:TDatetime` and `EndDate:TDatetime` properties to defined the length and the position of the progressbar.

Property	Description
<code>StartDate : TDateTime</code>	The start date of a progressbar.
<code>EndDate : TDateTime;</code>	The end date of a progressbar.
<code>Duration : TDateTime;</code>	(read only) The overall duration of the progressbar.

Also text can be added to the progressbar using the `TextHalo` object of the progressbar. The following table summarized the properties of the `TextHalo` object.

Property	Description
TextHalo.TextLeft:String	A text - left of the progressbar
TextHalo.TextTop:String	A text displayed above the progressbar
TextHalo.TextRight:String	A text – right of the progressbar.
TextHalo.TextBottom:String	A text displayed below the progressbar
TextHalo.TextCenter:String	A text displayed inside the progressbar.
Hint:String	A hint for the progressbar

For the rest of this chapter we will focus on properties that are defining the layout of the progressbar. In comparison the taskbars, progressbars do only have one solid fill draw style. If you want to change the color of the progressbar use the `ProgressBar.Settings.Color:TColor` property.

The other properties of the progress bars are listed in the following table:

Property	Description
Settings.Height	The height of the progressbar
Settings.Color	The color of the progressbar
Settings.BorderColor	The bordercolor of the progressbar

The following code example shows how to add a progressbar and change some of its properties. Please note to add the units `wgsProgressbar` and `wgsProgressBarSettings` to the uses clause of your delphi formular.

- First we create a new `TWGSProgressBar` and add it to the `GanttGraph` using the `BarAdd` method of the `DataSource`. After the bar is added we assign a text string to the bar and change the color.

```
(30) uses ...,
      wgsProgressbar,
      wgsProgressBarSettings;

...
procedure TForm1.AddProgressBar;
var
  progressbar : TWGSProgressBar;
begin
  progressbar := TWGSProgressBar.Create(now,now+4,1);
  WGSDataSource1.BarAdd(progressbar,1);
  progressbar.textHalo.TextRight := 'Progressbar';
  progressbar.Settings.Color := clRed;
  WSGGanttGraph1.RePaint;
end;
```

Milestonebars

Milestones represent a special date in your project so for example a deadline. They are visualized as a small glyph. The following example shows how to add a milestone and assign a picture to it.

- After creating a `MileStoneBar` object and adding it using the `AddBar` method of the `DataSource` we create a new `Picture` within the `PictureResource`.

```
(31) uses ...,
      wgsMilestone;

...
procedure TForm1.AddMileStoneBar;
var
  milestonebar : TWGSMileStoneBar;
begin
  milestonebar:= TWGSMileStoneBar.Create(Trunc(Now),Trunc(Now)+1,1);
  WGSDataSource1.BarAdd(milestonebar, 1);
  milestonebar.PictureResource.Picture := TBitmap.Create;
  milestonebar.PictureResource.Picture.LoadFromFile('D:\milestone.bmp');
end;
```


As already known from the other bars the milestone bar does also have the `TextHalo` object, providing the possibility to display text besides the milestone. The following table summarize all properties of the `TextHalo` object.

Property	Description
<code>TextHalo.TextLeft:String</code>	A text - left of the milestone
<code>TextHalo.TextTop:String</code>	A text displayed above the milestone
<code>TextHalo.TextRight:String</code>	A text – right of the milestone.
<code>TextHalo.TextBottom:String</code>	A text displayed below the milestone
<code>TextHalo.TextCenter:String</code>	A text displayed inside the milestone.
<code>Hint:String</code>	A hint for the milestone.

The other properties of the milestonebar are listed in the table below.

Property	Description
<code>Settings.BorderColor</code>	The bordercolor of the milestonebar.
<code>Settings.Color</code>	For milestones the color is without any functions.
<code>Settings.BorderThickness</code>	The thickness of the border of milestones.
<code>Settings.ImageFile</code>	Within the <code>ImageFile</code> property the developer can store any textual information, e.g. the original filename of the milestone, which he may use for internal program logic.

If you want to change the Transparency of the MileStone you have to access the `Picture` of the `PictureResource` object.

Property	Description
<code>PictureResource.Picture.Transparent</code>	The transparency of the MileStone glyph.

Imagebars

If you want to display a graphic within the `GanttGraph` you can use an imagebar. Please keep in mind that you are unable to display images (and milestones too) within the left part of the `GanttGraph` – the `TreeGrid`. If you want to know how to add an imagebar, please have a look at the following code snippet. Please add the unit `wgsImageBar` to the uses clause of your delphi formular.

- In this example we add an `ImageBar` to the `GanttGraph`. After adding the image we set its `AutoStretch` property to true. After that we change the length of the `ImageBar` to two days to resize the image.

```
(32) uses ...,
      wgsImageBar;

...
procedure TForm1.AddImageBar;
var
  ImageBar : TWGSImageBar;
begin
  ImageBar := TWGSImageBar.Create(Trunc(now), Trunc(Now)+3, 1);
  ImageBar.PictureResource.Picture := TBitmap.Create;
  ImageBar.PictureResource.Picture.LoadFromFile('D:\ppid11.bmp');
  WGSDataSource1.BarAdd(ImageBar, 1);
  ImageBar.AutoStretch := True;
  ImageBar.EndDate := Trunc(now)+2;
end;
```

Image bars do have – as all other types of bars – the `TextHalo` object allowing them to display a text beside them.

Property	Description
<code>TextHalo.TextLeft:String</code>	A text - left of the imagebar.
<code>TextHalo.TextTop:String</code>	A text displayed above the imagebar.
<code>TextHalo.TextRight:String</code>	A text – right of the imagebar.
<code>TextHalo.TextBottom:String</code>	A text displayed below the imagebar
<code>TextHalo.TextCenter:String</code>	A text displayed inside the imagebar.
<code>Hint:String</code>	A hint for the imagebar.

Other properties of Image bars are displayed in the list below:

Property	Description
Settings.BorderColor	The bordercolor of the imagebar.
Settings.Color	For imagebars the color is without any functions.
Settings.BorderThickness	The thickness of the border of imagebars.
Settings.ImageFile	Within the ImageFile property the developer can store any textual information, e.g. the original filename of the imagebar, which he may use for internal program logic.
AutoStretch	Specifies whether the Image is stretched into the boundaries defined by the start- and enddate.

Textbars

Textbars can be used to display larger text information within the GanttGraph as it is possible using the text properties of the TextHalo object. A textbar, as well as imagebars, can cover more than one row of the GanttGraph. When creating a textbar its horizontal dimensions are defined by the StartDate:TDateTime and the EndDate:TDateTime. To define the vertical dimension you have to use the Height:Integer property of the TextBar that specifies the height of a TextBar in pixel.

The following code example demonstrates how to create a textbar.

- As textbars are declared in the unit wgsTextBar, please add this unit to the uses clause of your Delphi formular. For assigning some text to the textbar, simply use the text:String property.

```
(33) uses ...,
      wgsTextBar;

...
procedure TForm1.AddImageBar;
var
  TextBar : TWGSTextBar;
begin
  TextBar:= TWGSTextBar.Create(Trunc(now), Trunc(Now)+ 2, 1);
  TextBar.Text := 'Hello World';
  TextBar.height := 50;
  WGSDataSource1.BarAdd(TextBar, 1);
  WSGanttGraph1.Repaint;
end;
```

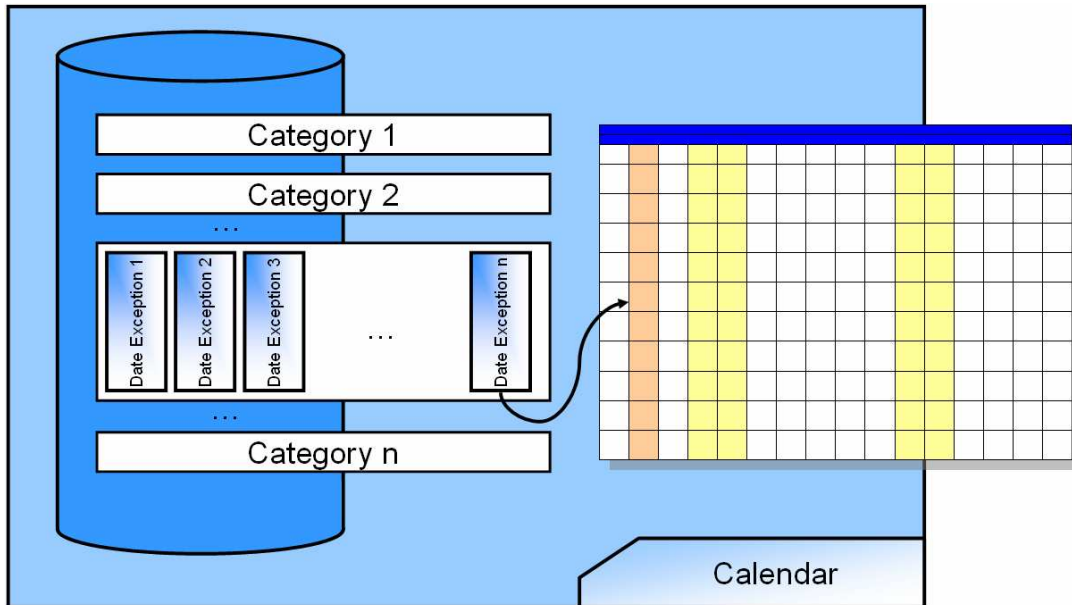
The Settings object of the textbar holds further properties:

Property	Description								
Settings.Font:TFont	Specifies the font of the textbar.								
Settings.Color:TColor	The background color of the textbar. Is applied only if the transparent flag is set to false.								
Settings.VAlign:Integer	The vertical alignment of the text. The following sub-table shows the possible values and their corresponding alignment. <table border="1" data-bbox="679 1659 924 1753"> <thead> <tr> <th>Value</th><th>Alignment</th></tr> </thead> <tbody> <tr> <td>0</td><td>Top</td></tr> <tr> <td>1</td><td>Bottom</td></tr> <tr> <td>2</td><td>Center</td></tr> </tbody> </table>	Value	Alignment	0	Top	1	Bottom	2	Center
Value	Alignment								
0	Top								
1	Bottom								
2	Center								
Settings.HAlign:Integer	The horizontal alignment of the text. The following sub-table shows the possible values and their corresponding alignment. <table border="1" data-bbox="679 1877 924 1971"> <thead> <tr> <th>Value</th><th>Alignment</th></tr> </thead> <tbody> <tr> <td>0</td><td>Left</td></tr> <tr> <td>1</td><td>Right</td></tr> <tr> <td>2</td><td>Center</td></tr> </tbody> </table>	Value	Alignment	0	Left	1	Right	2	Center
Value	Alignment								
0	Left								
1	Right								
2	Center								
Settings.Transparent:Boolean	The textbar will be displayed transparent if set to true.								

Calendar

The Gantt Control Component contains a 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 GanttGraph. Please note, that it is not possible to define different calendars for different rows.

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



As you can see the calendar consists of a list of (date)-categories. To each category, date exceptions can be added. There are different date category types, such as a weekend date category and there are different types of date exceptions, such as fixed, recurring date exceptions or date exceptions like Saturday and Sunday.

The appearance and the behaviour of a single date exception are mainly defined by the properties of its higher up date category. The date category specifies whether its date exceptions define working or non-working times. Also the visibility and the color used to visualize a date exception can be specified within the date category. The following table summarizes the properties of a date category.

Property / Methods	Description
Name:String	The name of the date category. Note: This name is just for the purpose of identification a date category.
Color:Boolean	The color that is used to visualize the date exceptions of this category.
Working:Boolean	Specifies whether the date exceptions of this category are handled as working dates.
DatesCount:Integer	Returns the number of dates that are defined within this date category.
Visible:Boolean	Specifies whether the date exceptions of this category should be displayed or not.
addDate(Date:IWGSDateException)	Adds a date exception to the date category
removeDate(Index:Integer)	Removes a date exception from the date category
matchDate(Date:TDateTime):IWGSDateException	Returns the date exception if there is a date exception for the given date, otherwise it will return NIL.

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

The calendar can be accessed by the DataSource's `calendar:TWGSDocumentCalendar` property. It provides useful methods and operations to manipulate date categories and to set the global time mode.

Property / Methods	Description
<code>IsWorking (Date:TDateTime) :Boolean</code>	Returns true if the specified date is defined as a working date.
<code>GetDateLabel (Date:TDateTime) :String</code>	Returns the label for the specified date, if there was found a matching entry for this date.
<code>MatchCategory (Date:TDateTime) :IWGSDateCategory</code>	If there was found a category for the specified date, this category will be returned, otherwise NIL will be returned.
<code>AddDateCategory (Category:IWGSDateCategory)</code>	Adds a new date category to the category list.
<code>Categories[] :IWGSDateCategory</code>	Accessible array of the date categories of the calendar object.
<code>CategoriesCount:Integer</code>	Returns the total number of categories
<code>RemoveDateCategory (Name:String)</code>	Removes a date category, specified by its name, from the date category list.
<code>MoveUp (Name:String)</code>	Moves up one position the specified date category in the date category list.
<code>MoveDown (Name:String)</code>	Moves down one position the specified date category in the date category list.
<code>SetTimeMode (TimeMode:TWGSTimeMode)</code>	Sets the time mode of the calendar. The time mode of the calendar describes the smallest time mode of all containing time scales.

- 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. Before you reproduce this code example, please add the unit `wgsDocumentCalendar` to the uses clause of your mainform.

```
(34) uses ...,
      wgsDocumentCalendar;

...
procedure TForm1.AddDateException;
var
  fixedDate : TWGSFixedDateException;
  ACategory : TWGSDateCategory;
begin
  // First we add a new date category ...
  ACategory := TWGSDateCategory.Create('A new date category', clLime);
  // ... after that, we create a fixed date exception
  fixedDate := TWGSFixedDateException.Create(Trunc(Now), 'A Date Exception');
  // Now we will add the date exception to the category
  ACategory.addDate(fixedDate);
  // Finally, we will add the category to the calendar.
  WGSDataSource1.Calendar.addDateCategory(ACategory);
  WSGanttGraph1.Repaint;
end;
```

Fixed date exceptions can be used to specify a single date/day. If you want to define recurring date exceptions you have to create a `TWGSRecurringDateException` date exception as shown in the example below.

- Please add the unit `wgsDocumentCalendar` and the unit `wgsCalendar` to the uses clause of your mainform.

```
(35) uses ...,
      wgsTextBar;

...
procedure TForm1.AddRecurringDateException;
var
  recurringDate : TWGSRecurringDateException;
  ACategory : TWGSDateCategory;
begin
  ACategory := TWGSDateCategory.Create('A new date category', clLime);
  recurringDate := TWGSRecurringDateException.Create(tmWeek, // a weekly date
    1, // total length of one day
    1, // starts from Monday
    2, // till Tuesday
    0, // from 0 a.m.
    0, // to 0 a.m.
    'team meeting', // the label
    Trunc(now)-100,
    Trunc(now)+100);
```

```

ACategory.addDate(recurringDate);
WGSDataSource1.Calendar.addDateCategory(ACategory);
WGSgantGraph1.Repaint;
end;

```

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 graph 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 visualisation and the calculation of the critical path can be enabled/disabled if you use the `autoUpdateCriticalPath: Boolean` property of the table. The color that is used for highlighting the critical path can be specified if you use the `criticalPathColor: TColor` property of the table.

Property	Description
<code>WGSgantGraph.Table.autoUpdateCriticalPath: Boolean</code>	Enables/Disabled the critical path.
<code>WGSgantGraph.Table.criticalPathColor: TColor</code>	The critical path color.

By default the critical path is enabled. Please note that after adding a new bar, a bar is 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 that the critical path is displayed within the gantt graph.

To decide whether an object is part of the critical path or not you can access the boolean property `criticalElement: Boolean` of the taskbar. The property is read- and writable, although it is recommended to not change the value as it is calculated internally by the gantt component itself.

Furthermore the critical path includes the calculation of the earliest possible start date, the earliest possible finish date, the latest possible start date and the latest possible end date without rescheduling any following bars. The following table summarizes those dates:

Property	Description
<code>FESTime: TDateTime</code>	A datetime property defining the earliest possible start time of a task/bar. The earliest start time depends on the connection ending at the bar. The task can not start earlier because other tasks have to be finished/started first.
<code>FLFTime: TDateTime</code>	A datetime property defining the latest possible finish time of a task/bar. The latest possible finish time of a task is calculated by the latest possible start time and its duration.
<code>FLBTime: TDateTime</code>	A datetime 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.

FGBTime:TDateTime	A datetime 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 endtime. If the global buffer is 0 then any delay of this bar/task would extends the overall project time. If the global buffer is zero, then the according task/bar is part of the critical path.
FEFTime:TDateTime	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.
FLSTime:TDateTime	A datetime property defining the latest possible start time of a task/bar. The latest starttime 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 by using the accessor methods of the `pertbar` object of a gantt row or if you access a task bar directly and typecast it as a `TWGSTaskbar` as shown below.

- Please note that the dates shown in the table above are *read only* values. The following code example assigns the earliest start of the second row to the variable `earliestStart:TDateTime`.

```
(36) ...
procedure TForm1.AccessDates;
var
    EarliestStart : TDateTime;
begin
    // option 1
    EarliestStart := WSGanttGraph1.Table.Rows[1].PertBar.FESTime;
    // option 2
    EarliestStart := TWGSTaskBar(WSGanttGraph1.Table.Rows[1].Bars[0]).FESTime;
end;
```

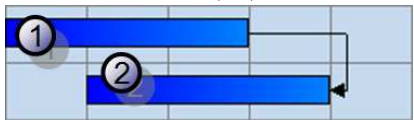
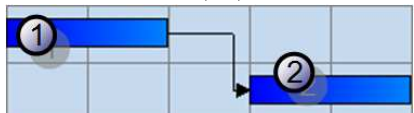
Connection between bars

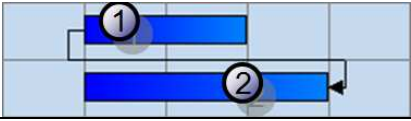
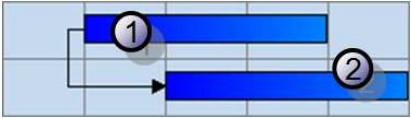
For modelling temporal and causal relationships between two tasks/activities you can connect two bars with each other and create a connection. Connections are represented as `TWGSBarConnection` objects and are defined in the unit `wgsBarConnection`.

There are two different ways to create a connection between two bars. First you can create a connection at runtime via the user-interface interaction. To do so, please move the mouse cursor over the start or the end of a taskbar, hold the [Shift] key down and press the left mouse button, moving 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, if you use the `ConnectionAdd(Connection:TWGSBarConnection)` method of the `DataSource` component.

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

Connection (Example)	Restriction	connection.setConnType(t)
Finish - Finish (FF) 	Task (#2) ends always after the end of task (#1).	$t=0$
Finish - Start (FS) 	Task (#2) starts always after the end of task (#1).	$t=1$

<p><i>Start - Finish (SF)</i></p> 	Task (#2) ends always after the start of task (#1)	$t=2$
<p><i>Start - Start (SS)</i></p> 	Task (#2) starts always after the start of task (#1)	$t=3$

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

Property	Description
<code>criticalElement: Boolean</code>	Indicates whether the connection is part of a critical path (see chapter critical path)
<code>minDistance: Integer</code>	The minimal distance between two bars in days.
<code>distanceFixed: Boolean</code>	Specifies whether the distance between the two connected bars should stay fix when scheduling one of the connected bars.
<code>Duration: Integer (ReadOnly)</code>	The distance between two bars.
<code>Origin : TWGSGanttBar</code>	The gantt bar where the connection starts from.
<code>Target : TWGSGanttBar</code>	The gantt bar where the connection ends.

- The following code example adds 3 new rows and 2 new bars to the gantt graph and connect the bars with each other. When adding a new connection you have to assign the bar, where the connection starts from (origin) and the bar the connection is connected to (target).

After adding a connection its recommended to call the `WGSDatasource.ResolveDistanceViolations(fixed: TWGSGanttBar)` operation, to reschedule the connected bar if necessary. Note: The bar specified by the parameter `fixed` will not be rescheduled. Instead the bar that is connected with `fixed` will be rescheduled to fit the restrictions of the corresponding connection based upon its connection type.

```
(37) uses ...,
      wgsBarConnection,
      wgsTaskBar;
...
procedure TForm1.CreateAndConnectBars(Sender: TObject);
var
  i      : Integer;
  connection : TWGSBarConnection;
  bar1, bar2 : TWGSTaskBar;
begin
  // First we add some new rows, ...
  for i := 1 to 3 do WGSDatasource1.RowAppend;
  // ..., then we create and add two new taskbars
  bar1 := TWGSTaskBar.Create(Trunc(now), Trunc(now)+3,0);
  bar2 := TWGSTaskBar.Create(Trunc(now)+1, Trunc(now)+4,2);
  WGSDatasource1.BarAdd(bar1, 0);
  WGSDatasource1.BarAdd(bar2, 2);
  // After that, we create a new TWGSBarConnection object
  connection := TWGSBarConnection.Create;
  connection.origin := WGSanttGraph1.table.Rows[0].Bars[0];
  connection.target := WGSanttGraph1.table.Rows[2].Bars[0];
  connection.setConnType(3);
  // and add the connection to the DataSource
  WGSDatasource1.ConnectionAdd(connection);
  WGSDatasource1.ResolveDistanceViolations(bar1);
  WGSanttGraph1.Repaint;
end;
```

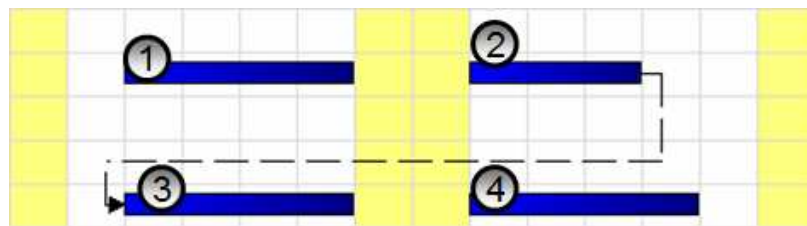
For accessing single connections use the connections array of the document object (`WGSanttGraph.document.connection[n]`). The `Document: TWGSGanttDocument` holds further useful constructions for accessing connections, see the following table.

Function (WGSanttGraph.Document)	Description
<code>countConnections: integer</code>	Returns the number of connections.

<code>findConnection(bar1:TWGSGanttBar, bar2:TWGSGanttBar; [connType:Integer=-1])</code>	Returns the connection for the specified bars. If there was no connection found it will return NIL.
<code>Connection[]:TWGSBarConnection</code>	Accessible array of all connections.

If you want to delete an existing connection, you have to use the `ConnectionDelete(Connection:TWGSBarConnection):Boolean` command of the `DataSource`.

By default a connection will reschedule the bar, where the connection points to - according to the temporal relationship defined by the type of the connection. So for example the target bar of a Finish-Start (FS) connection will be rescheduled so that the target bar starts always after the origin bar ends. However, in some cases it is not possible to reschedule the target bar as there are some other bars so that there is no more space for the target bar to be rescheduled. If so - the Gantt Control Component tries to reschedule the origin bar to fit the needs of the temporal relationship. If this is also not possible the connection is "invalid". Please have a look at the following graphic, for an example of an invalid connection.



As you can see above, the bar (#2) can not start earlier as there is already the bar (#1). Also the bar (#3) can not start directly after the bar (#2) ends, as there is already the bar (#4). Invalid connections are always displayed by a dashed line.

WGSDataSource

The DataSource component provides methods and functionalities for modifying the data of the gantt/pert chart. The GanttGraph, PertGraph and the KATPrintingPreview must be linked to the DataSource component. All linked component share the same data source.

The DataSource holds a reference to the tree (TWGSTreeGrid) and the table (TWGSGanttTable) of the GanttGraph that is linked to the DataSource. Furthermore the DataSource puts for every data-manipulating operation a corresponding undo operation on an undo-stack that enables the developer to undo any changes made to the data. Note: Changes made to the layout of the objects, e.g. font size, color etc. are not stored within the undo stack.

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

Function	Description
BarResize	Resizes a specified bar except pert bars. The range the bar should be resized is specified by the parameters DeltaX and DeltaY.
BarDelete	Deletes the specified bar
BarMove	Moves a specified bar to the given row and the given time range.
BarAdd	Adds a new bar to the gantt graph.
ConnectionDelete	Deletes a connection between two bar objects.
ConnectionAdd	Adds a new connection between two bar objects.
ColumnDelete	Deletes a column of the tree grid.
ColumnAdd	Adds a new column to the tree grid at the specified position.
ColumnAppend	Appends a new column at the end of the tree grid
RowAppend	Appends a new row to the gantt graph.
BranchDelete	Deletes an entire branch (all child rows) of an existing row.
RowDelete	Deletes a specified row.
DeleteSelected	Deletes all the rows that are currently selected within the tree grid.
RowClear	Clears the content of a row (all cell values and all corresponding bars).
RowChildInsert	Inserts a new child row for the specified row.
RowInsert	Inserts a new row.
RowsGroup	Groups a range of rows and increases their hierarchical nesting level
RowsChangeParent	Changes the parent row for a range of rows.
PertBarAdd	Adds a pert bar to the pert graph.
PertBarDelete	Deletes a pert bar from the pert graph.
PertConnectionAdd	Adds a new connection between two pert bars.
PertConnectionDelete	Deletes an existing connection between two pert bars.
ResolveDistanceViolations	If there is a restrictive connection including a fixed distance between two bars or a minimal distance between two bars – ResolveDistanceViolation tries to solve all logical violations for a specified bar. When resolving a distance violation to a bar, other connected bars are moved/resized this way, that the violation is eliminated.
ClearUndoReDoStack	Flushes the redo/undo stack.

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

Reference	Class	Unit
UndoReDoStack	TWGSUndoRedoStack	wgsUndoRedoStack
Tree	TWGSTreeGrid	wgsTreeGrid
Table	TWGSGanttTable	wgsGanttTable
Calendar	TWGSDocumentCalendar	wgsDocumentCalendar
Legend	TWGSLegend	wgsLegend
Root	TWGSganttDocument	wgsGanttDocument

UndoRedo Stack

The function **Undo: Boolean** and **ReDo: Boolean** will undo or redo any changes made to the DataSource. Any operation making changes to the data – whether it is initiated by the user interface, so when the end user drags in new objects or performed by program logic, using operation the DataSource provides, will be stored to the undo and redo stack.

- The following code sample shows how to use the undo / redo function

```
(38) ...
    // Undo
    WGSDataSource1.UndoRedoStack.Undo();
...
    // Redo
    WGSDataSource1.UndoRedoStack.Redo();
...
```

By default any single operation will be pushed to the undo- and redo stack. You can also group a set of operations so that it would be undone in one single step.

- For grouping a set of operations use the **StartGrouping** and **EndGrouping** statement. When calling the Undo() method all operations between those two statements will be undone or redone in one single step.

```
(39) ...
    // Creating a grouped stack of operations
    WGSDataSource1.UndoRedoStack.StartGrouping;
...
    // Perform operations that should be grouped here
...
    WGSDataSource1.UndoRedoStack.EndGrouping;
...
```

The following table summarizes all functions/methods of the undo and redo stack.

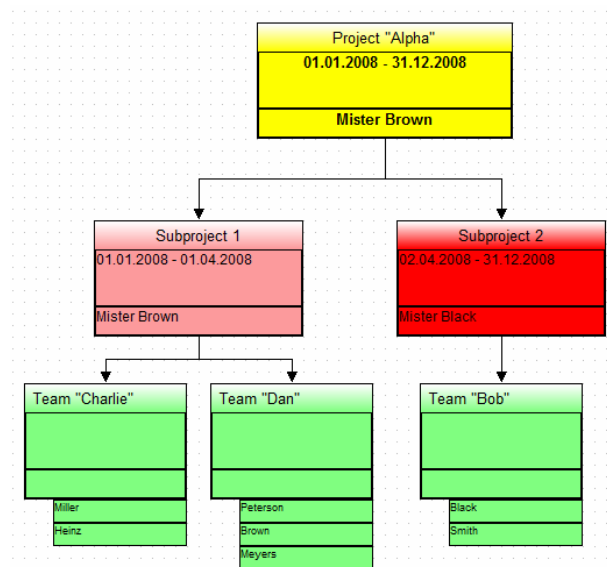
Function	Description
Undo():Boolean	Undo will undo the changes made to the data by the last operation – or by the last grouped stack of operations.
Redo():Boolean	If you have previously performed an undo operation all changes will be redone.
Clear	Clears the undo redo stack
StartGrouping	Begins a new group of operations that will be undone / redone in one step
EndGrouping	Finishes a group of operations that will be undone / redone in one step
UndoOperationCount:integer	Returns the total number of operations that can be undone.
RedoOperationCount:integer	Returns the total number of operations that can be redone.

WGSPertGraph

The PertGraph is an interactive user interface component that contains a tree grid and a pert chart. Unlike the GanttGraph the PertGraph 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.

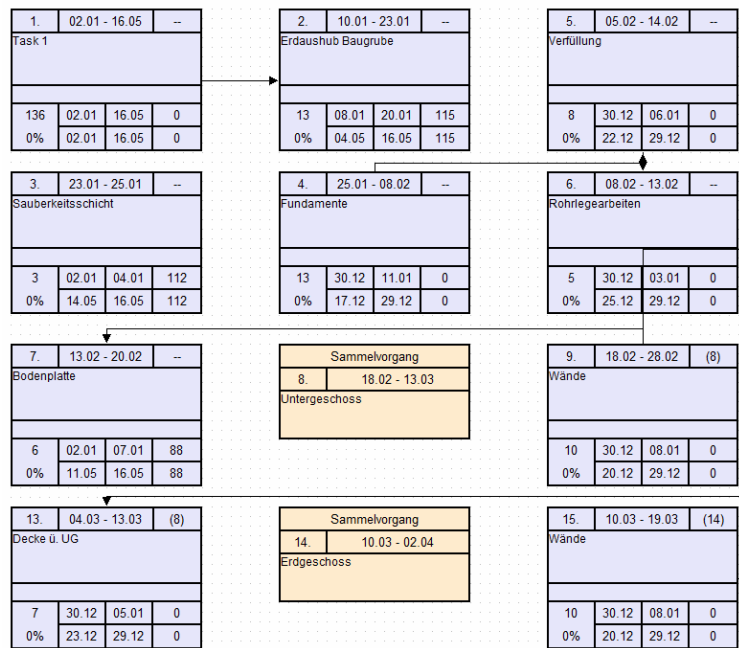
Within the PertGraph, as a part of the Gantt Control Software component, the duration of a task bar as well as the earliest start date, the earliest end date, the latest start date and the latest end date are represented as pert bars.

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 project aims, sub ordinate project aims and work packages can be visualized and modelled as elements of the pert chart. For example pert bars that represent work packages can be extended by a list of responsible employees. For modelling relations between aims and working packages the pert chart supports connecting pert bars with each other. Tasks of a gantt chart and a pert chart can be assigned to a unique datasource. This offers the advantage that changes made to tasks in a gantt chart are automatically updated within the pert chart. Please have a look of the following picture for an example of hierarchical project structuring using pert bars.



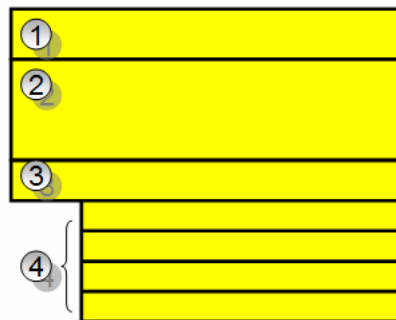
Note: All pert bars shown (referred as type I) in the example above do not display any critical time requirements. Therefore the `PertType` property of the pert bars is set to `wgspbtNoTimeSpan`.

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 assign `wgspbtTimeSpan` for the `PertType` (referred as type II). The picture below shows pert bars displaying additional time span information.



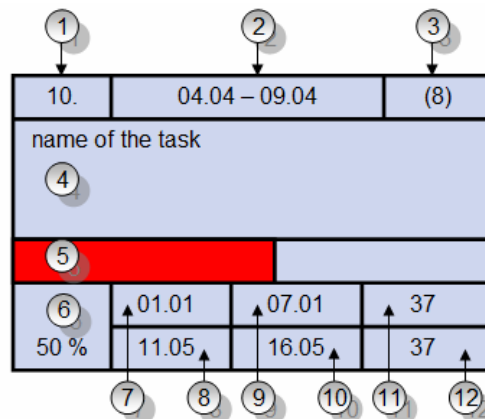
Pert bar (Type I – “No Time Span”)

The pert bar type I, that's `PertType` is set to `wgspbtNoTimeSpan`, is divided into 4 different segments. These are (1) `TopSection`, (2) `MainSection`, (3) `BottomSection` and (4) `AdditionalSections`.



Pert bar (Type II – “Time Span”)

The pert bar type II, that's `PertType` is set to `wgspbtTimeSpan`, displays additional time span information as shown in the picture below as well as the task name, the 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 visualisation of the progress of the pert bar.
- 6 The numeric value of the pert bar's 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.

Pert bar properties

The following table summarizes the properties defined for the pert bars.

Property	Description	Pert Bar Type
Settings:TWGSPertBarSettings	A reference to the common settings object of pert bars.	I, II
PertType:TWGSPertBarType	The pert bar type (wgsptNoTimeSpan, wgsptTimeSpan) of the pert bar.	I, II
Detailed:Boolean	If detailed is set to true the additional sections of the pert bar will be displayed.	I
MainSection:TWGSPertBarSection	A read- and writeable property defining the TWGSPertBarSection object, representing the main (middle) section of the pert bar.	I
TopSection:TWGSPertBarSection	A read- and writeable property defining the TWGSPertBarSection object, representing the top section of the pert bar.	I
BottomSection: TWGSPertBarSection	A read- and writeable property defining the TWGSPertBarSection object, representing the bottom section of the pert bar.	I
AdditionalSections : TObjectList	An object list holding additional sections that are displayed beneath the pert bar, if Detailed is set to true.	I
X:Integer	The x-coordinate of the pert bar.	I,II
Y:Integer	The y-coordinate of the pert bar.	I,II
UserDefinedNumber:String	The value used for displaying the number of the pert bar can be user defined,	II

The Settings:TWGSPertBarSettings object, that is defined in the unit wgsPertBarSettings of the pert bar provides the following properties:

Property	Description
BorderColor:TColor	The border color of the pert bar
BackgroundColor:TColor	The background color of the pert bar
BackgroundSummaryColor:TColor	The background color, if the pert bar visualizes a summary bar.
BorderThickness:TColor	The thickness in pixel of the pert bar border.
Height:Integer	The height of the pert bar in pixel.
Width:Integer	The width of the pert bar in pixel.

The object TWGSPertBarSection, implemented in the unit wgsPertBarSection, declares following properties:

Property	Description
Settings:TWGSTextSettings	Settings for the text that is displayed inside the pert bar section.
Settings.Font:TFont	The font object for the pert bar section.
Settings.Color:TColor	The color value for the text.
Settings.VAlign:Integer	The vertical alignment of the text
Settings.HAlign:Integer	The horizontal alignment of the text
Text:String	The text that is displayed inside the pert bar section.

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 linked a PertGraph to a datasource, a pert bar will automatically be created if you (or the enduser) add a taskbar to the datasource. Also whenever you add a pert bar by programm logic or via the user interface a corresponding task 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 pertbar 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. By default all new pert bars share the settings that are defined in the `defaultPertBarSettings` object of the table (`WGSPertGraph.table.defaultPertBarSettings`).

When adding a pert bar per program code you can use the `PertBarAdd(Row:integer;X,Y:Integer)` method of the datasource. As mentioned above you can also add a taskbar – this will automatically create a pert bar for the given row.

The code snippet below shows how to add a pert bar to a pert graph.

- To reproduce this, please add a new `WGSPertGraph` and a `WGSDatasource` component to your main form and set the `DataSource` property of the `WGSPertGraph` to the added `WGSDatasource` component. After that add the unit `wgsPertBar` to the use clause of your main form. For the example shown a pert bar will be added to the first row. If you want to add pert bars to other rows you first have to append new rows using the `RowAppend()` command.

```
(40) uses ... ,
      wgsPertBar;
...
procedure TForm1.AddAPertBar;
begin
  WGSDatasource1.PertBarAdd(0, 10,10);
  WGSPertGraph1.Repaint;
end;
```

Accessing pert bars

If you intend to access an existing pert bar you can use the `PertBar:TWGSPertBar` object of a `GanttRow:TWGSGanttRow`.

- The following code example extends the previous code. After accessing the pert bar of the first row its pert bar type is changed to `wgsptbNoTimeSpan`.

```
(41) uses ... ,
      wgsPertBar;
...
procedure TForm1.AddAPertBar;
begin
  WGSDatasource1.PertBarAdd(0, 10,10);
  WGSPertGraph1.table.Rows[0].pertBar.PertType := wgsptbNoTimeSpan;
  WGSPertGraph1.Repaint;
end;
```

Deleting pert bars

For deleting pert bars, use the `PertBarDelete()` statement as shown in the example below.

- First we ensure that there is a pert bar in the first row. If so we use the `PertBarDelete()` command to delete this pert bar. It is recommended to first use the prior code example to create a pert bar before you try to reconstruct the code snippet provided here.

```
(42) uses ...,
      wgsPertBar;
...
procedure TForm1.AddAPertBar;
begin
  if WGSPertGraph1.table.Rows[0].pertBar <> nil then
    WGSDataSource1.PertBarDelete(WGSPertGraph1.table.Rows[0].pertBar);
    WGSPertGraph1.Repaint;
end;
```

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](#)).

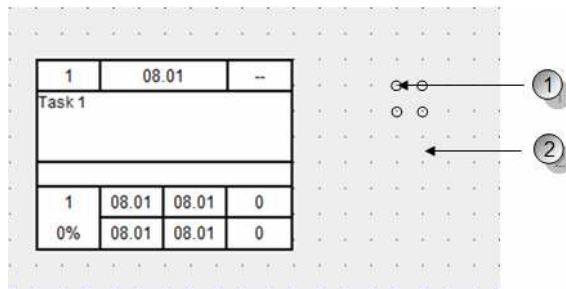
You can directly add two pert bars by using the `PertConnectionAdd()` statement. How this can be done is shown in the next code example.

- We will create a connection here. To connect the pert bar of the first row to the pert bar of the second row. So the following example will work only, if you already have created pert bars for the first two rows. To properly create a `TWGSBarConnection` object you have to add the unit `wgsBarConnection`.

```
(43) uses ...,
      wgsPertBar,
      wgsBarConnection;
...
procedure TForm1.ConnectTwoPertBars;
var
  Con : TWGSBarConnection;
begin
  Con := TWGSBarConnection.Create;
  Con.Parent := WGSPertGraph1.document;
  Con.origin := WGSDataSource1.Table.Rows[0].pertBar;
  Con.target := WGSDataSource1.Table.Rows[1].pertBar;
  Con.setConnType(0);
  WGSDataSource1.PertConnectionAdd(Con);
  WGSPertGraph1.Repaint;
end;
```

Formatting the Pert Graph

To support a better handling when moving objects the PertGraph provides a raster (1). The size of the raster can be adjusted by using the `PertGridSize:TPoint` property as shown below



- The x-coordinate of the point object is used to specify the width and the y-coordinate the height of the raster.

```
(44) ...  
procedure TForm1.ConnectTwoPertBars;  
begin  
    WSPertGraph1.table.pertGridSize := Point(30,30);  
end;
```

If you want to change the back ground color of the PertGraph (2) you have to use the `pertGraphBackColor:TColor` property of the table. The following table summarizes all properties referring to change the layout of the Pert Graph.

Property	Description
Table.PertGridSize:TPoint	Specifies the width and the height of the pert raster.
Table.PertGraphBackColor:TColor	The background color of the PertGraph.

KaTPrintPreview

In the final stage of the workflow process it is often required to present and print out the created chart diagrams. A flexible printing preview that is highly adaptable to the users 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 Printing 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, please have a look at the chapter (-> Localization).

To set up the printing preview, please place a new KATPrintPreview component on your delphi formular that contains a GanttGraph or a PertGraph and a DataSource component. After that, you have to connect the KATPrintPreview component to the DataSource.

To do so, please select the KATPrintPreview component in the Delphi Object Inspector (F11) and assign WGSDataSource1 for the DataSource property.

Before the printing preview is invoked or before a chart diagram is directly printed out, we have to setup the printer and its format. At this point it is important to know that the end-user is unable to set-up the printer and the format in the printing preview – so this must always be done before showing the printing preview.

There are different approaches to select the correct printer. To select a printer we can use the Delphis printer object, and assign a value for the printer index as shown in the code example below:

- Please add the delphi unit `printers`, before you reproduce this code.

```
(45) ...  
procedure TForm1.SetUpAPrinter;  
begin  
    Printer.PrinterIndex := 0;  
end;
```

A much better way, would be to let the end user decide on which printer device the chart diagrams should be printed. To do so, we just can use a `PrinterSetUpDialog` that Delphi already provides for us, or we use the encapsulated `PrinterSetUpDialog` of the printing preview as shown here:

- Here we show how to invoke the printer setup dialog of the printing preview.

```
(46) ...  
procedure TForm1.SetUpAPrinter;  
begin  
    KaTPrintPreview1.ExecutePrinterSetUpDialog;  
end;
```

Now it is important to specify the content that should be printed. After that the printing preview can be shown or the content can be printed.

- Please keep in mind, to always specify the content that should be printed before invoking the preview.


```

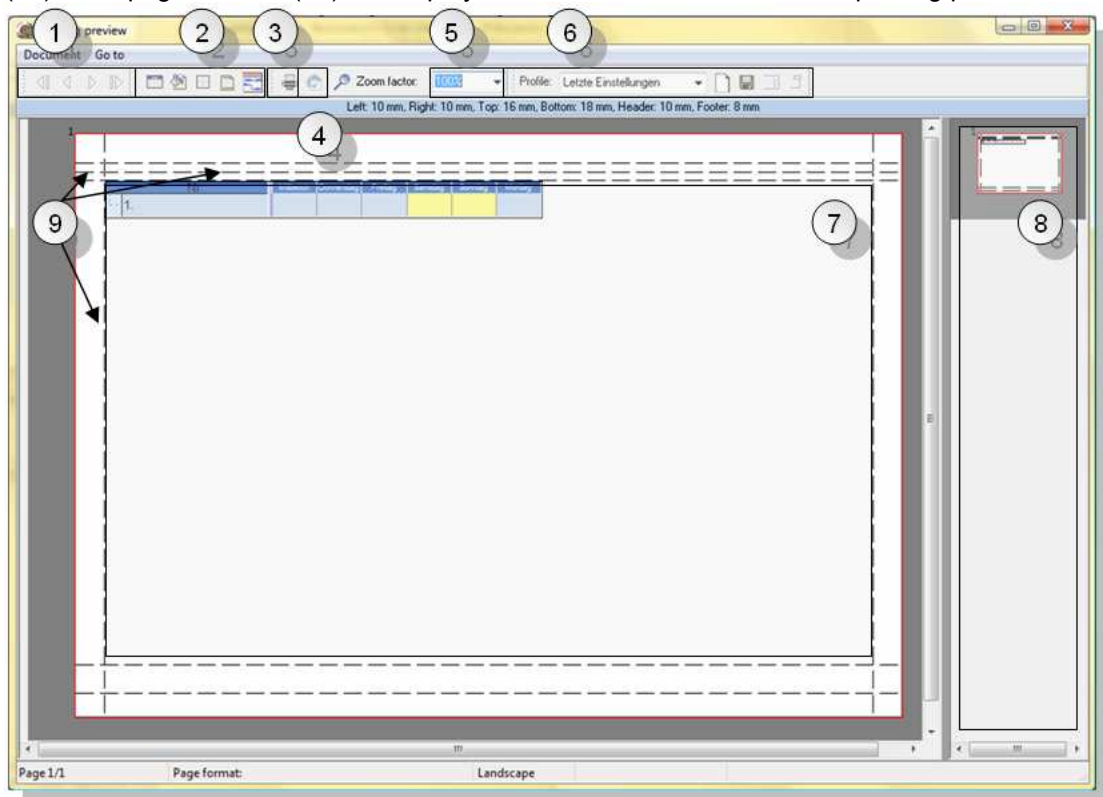
(47) ...
procedure TForm1.DoPrinting;
begin
    // Both, the gantt graph and the tree grid should be printed
    KaTPrintPreview1.PrintOptions.PrintGraph := true;
    KaTPrintPreview1.PrintOptions.PrintTable := true;
    // Show the printing preview.
    KaTPrintPreview1.DoPrint(true, false);
end;

```

The `DoPrint` method expects two Boolean parameters. The first specifies whether the printing preview should be shown (`true`) or whether the charts should be directly printed without showing a preview (`false`). If the second Boolean parameter is set to `true`, the “page setup dialog” of the printing preview is shown.

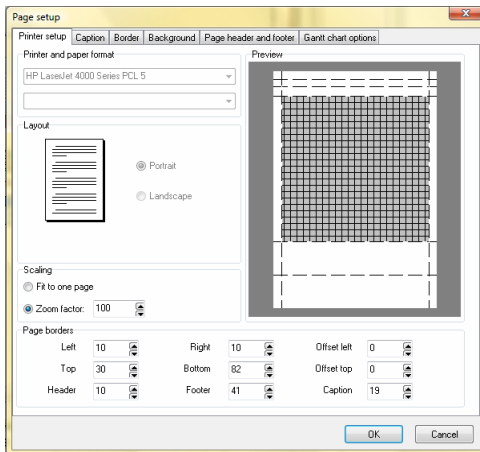
Printing Preview

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 area (#2) are used to set up and format the content of the printing 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 button shows a sub dialog, where the user can set up layout options according to the buttons category. The tool button (#3) is used to print the content of the printing 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 result. The printing 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 printing 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 printing preview.



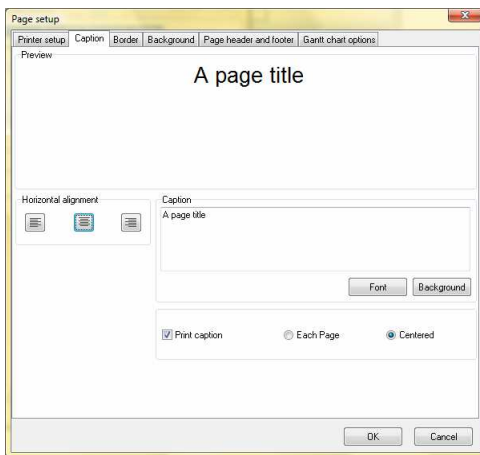
Each page border can be dragged with the mouse. The area that is used for the page header the page and the page footer is defined by its border. Please note, that whenever the area is too small to display all of its content - it will be displayed using a dash style.

Page Setup dialog

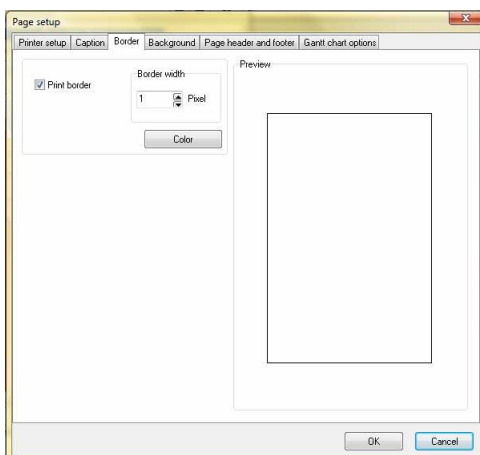


Here you can see the selected printer, the format and its orientation. Also you can specify whether the content should be stretched to fit one page (*Fit to page*) or if a zoom factor should be applied (*Zoom factor*) when printing. Please note, that this zoom factor affects the resulting print-out unlike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

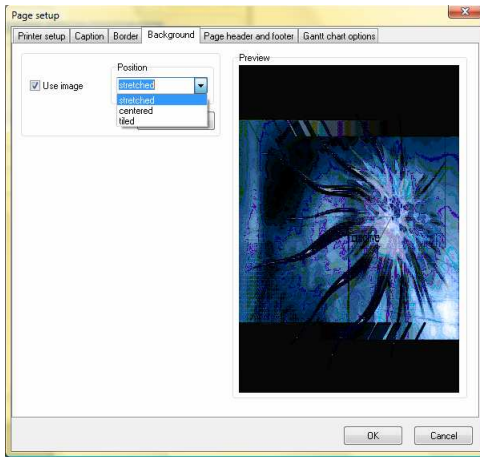
Also the user can specify the page borders at the bottom of the dialog. It is also possible to set up a global printing offset that shifts the print-out according to the specified values to the right and/or to the bottom.



The user is able to apply a page title (caption). The page title is displayed beneath the page header. A background color, the font and the alignment can be adjusted for the page title. Furthermore the user can specify whether the page title should be printed for each page or not.

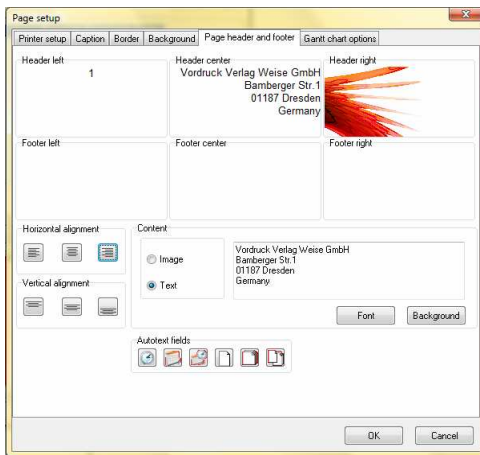


It's possible to print a border that surrounds the entire content. A color and the width of the border can be specified here.



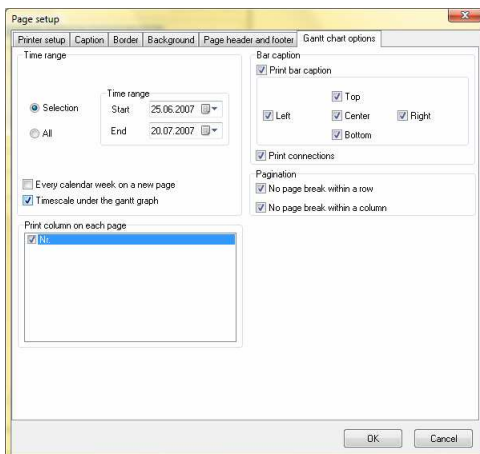
Here the user can define a background for the gantt chart or pert chart diagram. A background can be drawn by using a solid fill background defined by its color.

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, centred and tiled).



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.



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 is more than one page).

PrintOptions

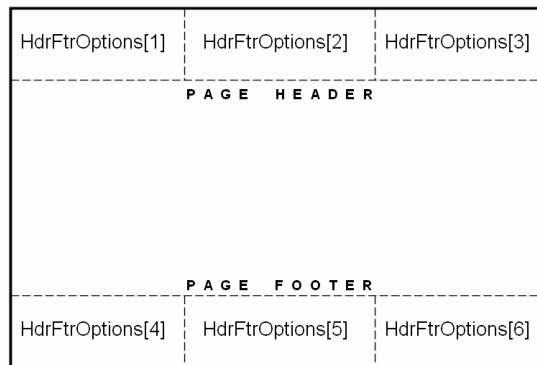
The `KaTPrintPreview` component provides the `PrintOptions` object that allows access to the most relevant print options shown above. The following list describes the properties of the `PrintOptions` object.

Property	Description
<code>PrintJobTitle:String</code>	The name of the print process.
<code>BkIsImage:Boolean</code>	Specifies whether an image is used for the background.
<code>BkFileName:String</code>	The filename of the image that is used as the background.
<code>BkColor:TColor</code>	The background color
<code>BkStyle:Integer</code>	Specifies how the background image is drawn.
	Value Draw Mode

	<table> <tr><td>0</td><td><i>stretched</i></td></tr> <tr><td>1</td><td><i>centred</i></td></tr> <tr><td>2</td><td><i>tiled</i></td></tr> </table>	0	<i>stretched</i>	1	<i>centred</i>	2	<i>tiled</i>
0	<i>stretched</i>						
1	<i>centred</i>						
2	<i>tiled</i>						
PageMarginLeft:Integer	The left page margin in millimetres						
PageMarginRight:Integer	The right page margin in millimetres						
PageMarginTop:Integer	The top page margin in millimetres						
PageMarginBottom:Integer	The bottom page margin in millimetres						
PageMarginFooter:Integer	The footer margin in millimetres						
PageMarginHeader:Integer	The header margin in millimetres						
PageMarginTitle:Integer	The title margin in millimetres						
PageOffsetX:Integer	The left page offset in millimetres						
PageOffsetY:Integer	The top page offset in millimetres						
PageWidth:Integer	The total page width						
PageHeight:Integer	The total page height						
PrintTitle:Boolean	Specifies whether the page title (caption) should be printed						
PrintLegend:Boolean	Specifies whether a legend should be printed above the gantt/pert chart						
PrintGraph:Boolean	Specifies whether the gantt / pert chart should be printed						
PrintTable:Boolean	Specifies whether the tree grid should be printed						
PrintBarText:Boolean	Specifies whether the bar text captions should be printed						
PrintBarTextLeft:Boolean	Specifies whether the left bar text caption should be printed						
PrintBarTextRight:Boolean	Specifies whether the right bar text caption should be printed						
PrintBarTextTop:Boolean	Specifies whether the top bar text caption should be printed						
PrintBarTextBottom:Boolean	Specifies whether the bottom bar text caption should be printed						
PrintBarTextCenter:Boolean	Specifies whether the centre bar text caption should be printed						
PrintConnections:Boolean	Specifies whether the bar connections should be printed or not						
FitWeeks:Boolean	If the gantt chart is in week mode, then the weeks are stretched to a page. (This is only applied when the content is printed on several pages)						
FitToPage:Boolean	If set to true the gantt chart is stretched to fit one page						
ZoomFactor:Real	Assign a zoom factor ((>0)..1) that is used for the preview						
ManualZoom:Real	Assign a numeric zoom value (%) to zoom the content of the print preview						
TitleEachPage:Boolean	Specifies whether the title should be printed on each page						
TitleShow:Boolean	Specifies whether the title should be shown or not						
PixelFormat:TPixelFormat	The Pixelformat (pfDevice, pf1bit, pf4bit, pf8bit, pf15bit, pf16bit, pf24bit, pf32bit, pfCustom) used for the preview.						
BorderSize:Integer	If a border is printed surround the gantt/pert chart, border size determines the width in pixel of the border.						
BorderShow:Boolean	Specifies whether a surrounding border is shown or not.						
BorderColor:TColor	Specifies the color of the border.						
PageBreak:Boolean	If set to true, the printing preview tries to avoid a horizontal splitting of rows.						
ColumnBreak:Boolean	If set to true the printing preview tries to avoid a vertical splitting of columns.						
PrintTimeScaleUnder:Boolean	Specifies whether a time scale is printed beneath the gantt graph.						
StartDate:TDateTime	Specifies the start date of the time range for the printed gantt graph.						
EndDate:TDateTime	Specifies the end date of the time range for the printed gantt graph.						
TimeRangeAuto:Boolean	If set to true the time range of the gantt graph, used for printing, is determined automatically by its content.						
TimeRangeLeftOffset:Integer	If an automatic time range is applied (TimeRangeAuto). TimeRangeLeftOffset specifies the left time buffer in days.						
TimeRangeRightOffset:Integer	If an automatic time range is applied (TimeRangeAuto) TimeRangeRightOffset specifies the right time buffer in days.						
GraphKind:TKaTGraphKind	If a dataset is connected to a pertgraph and a ganttgraph, you can specify here what type of graph should be printed.						

Value	<i>printed graph</i>
katgkGantt	<i>Gantt Graph</i>
katgkPert	<i>Pert Graph</i>

The `PrintOptions` also provides the `HdrFtrOptions` array that allows access to the page header and footer. Starting from the top left corner the `HdrFtrOptions` array is consecutively numbered from `HdrFtrOptions[1]` to `HdrFtrOptions[6]` as shown in the picture below.



The HdrFtrOptions provides the following properties:

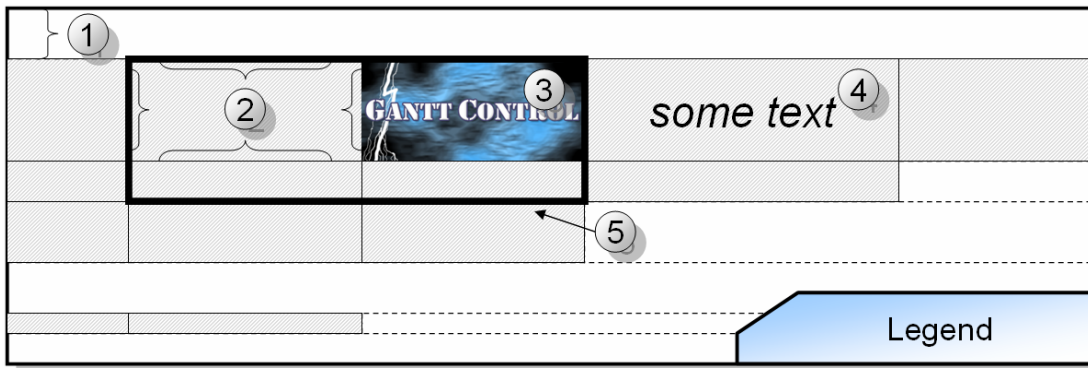
Property	Description								
isImage:Boolean	Specifies whether an image is used for the current segment.								
imageFile:String	Specifies the image file that is used for the current segment.								
imageStyle:Integer	Specifies the way the image is drawn within the current segment.								
	<table border="1"> <thead> <tr> <th>Value</th><th>Draw mode</th></tr> </thead> <tbody> <tr> <td>0</td><td>Stretched</td></tr> <tr> <td>1</td><td>Adjusted (normal)</td></tr> <tr> <td>2</td><td>Tiled</td></tr> </tbody> </table>	Value	Draw mode	0	Stretched	1	Adjusted (normal)	2	Tiled
Value	Draw mode								
0	Stretched								
1	Adjusted (normal)								
2	Tiled								
Valign:integer	The vertical alignment of the text or the image used for the current segment.								
HAlign:Integer	The horizontal alignment of the text or the image used for the current segment.								
Font:TFont	The font object.								
Contents:String	The text that is displayed within the segment, if isImage was set to false.								
Color:TColor	The background color for the segment.								

The PrintOptions holds also the possibility to format the title. The TitleOptions:TKaTPageTitleOptions object provides the following properties:

Property	Description								
Font:TFont	The font object for the title.								
Contents:String	The text that should be displayed as the title.								
Color:TColor	The background color for the title area.								
HAlign:Integer	The vertical alignment of the text. The following sub-table shows the possible values and their corresponding alignment.								
	<table border="1"> <thead> <tr> <th>Value</th><th>Alignment</th></tr> </thead> <tbody> <tr> <td>0</td><td>Left</td></tr> <tr> <td>1</td><td>Right</td></tr> <tr> <td>2</td><td>Center</td></tr> </tbody> </table>	Value	Alignment	0	Left	1	Right	2	Center
Value	Alignment								
0	Left								
1	Right								
2	Center								

Legend

To display any textual und graphical information you can create a legend. The legend will be displayed below the Ganttgraph of 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 interface provides access to the cells canvas – so it's possible to draw some images (#3) to a cell or to draw some textual information (#4). Please note that a row does not need to have cells at all, also each row can own a different amount of legend cells. Furthermore there is the possibility to assign a different horizontal alignment for each row. Also you can define borders (#5) surrounding cells. A border can be applied to a range of legend cells that is defined by the start row, the start cell, the end row and the end cell.

The following table shows the most relevant operations and properties of the `legend`. The `Legend` object can be accessed by the `DataSource`.

Property	Description
<code>rowCount:Integer</code>	The total amount of all rows of the legend.
<code>bordersCount:Integer</code>	The total amount of borders.
<code>addBlankRow(height:Integer; [n:Integer=1])</code>	Adds a new row to the legend.
<code>addBorder (border:TWGSLegendBorder)</code>	Adds a new border to the legend.
<code>getRowStartY (row:TWGSLegendRow)</code>	Returns the y start position of the specified row.
<code>getRowIndex (row:TWGSLegendRow) :Integer</code>	Returns the index of the specified row.
<code>clearRows</code>	Flushes all rows of the legend.
<code>clearBorders</code>	Flushes all borders of the legend.
<code>backGroundColor:TColor</code>	The background color of the legend
<code>Rows [] :TWGSLegendRow</code>	Accessible array of rows of the legend.
<code>BorderColor:TColor</code>	The color of the border
<code>BorderThickness:Integer</code>	The width of the border in pixel.
<code>Borders [] :TWGSLegendBorder</code>	Accessible array of the borders.
<code>DefaultTextSettings:TWGSTextSettings;</code>	The default text settings of the border cells.

The `rows` objects provide the following properties and functions:

Property	Description
<code>getCell (I:Integer) :TWGSLegendCell</code>	Returns the specified legend cell of a row.
<code>cellsCount:Integer</code>	Returns the total amount of cells of a row.
<code>addCell (Cell:TWGSLegendCell)</code>	Adds a new cell to the row.
<code>RowIndex:Integer</code>	Returns the row index of the current row.
<code>Length ([StartCell:Integer=0])</code>	Returns the total width of all cells of the row. If the parameter <code>StartCell</code> was defined. The function returns the summation of the rows width started from the <code>StartCell</code> cell.
<code>getCellRect (i:Integer) :TRect</code>	Returns the visible area of the specified cell.
<code>Cells [] :TWGSLegendCell</code>	Accessible array of the cells of a row
<code>Height:Integer</code>	Returns the height of the row.
<code>Alignment:WGSRowAlignment</code>	The horizontal alignment of all cells within this row

Value	Alignment
<code>raLeft</code>	Left justify
<code>raRight</code>	Right justify
<code>raCenter</code>	Center
<code>raStretch</code>	The cells are splitted specified by the <code>SplitAt</code> position

`SplitAt:Integer`

All cells that's ID is lesser or equal to the specified cell id are displayed left aligned the rest of the cell is right aligned. You have to

The cells object of legend rows provides the following properties and functions:

Property	Description
Width:Integer	The width of in pixel of this cell.
TextSettings:TWGSTextSettings	The text settings object of this cell.
Text:String	The text string that is displayed in this cell.
TextSettings.Color:TColor	Specifies the background color of this cell.

- The following delphi code shows how to add a new row to the legend, create a legend cell and add this cell to the row. After that the printing preview will be shown in such a way that the legend will become visible. To reproduce this code example, please add a KaTPrintPreview component to your delphi formular and link the KaTPrintPreview to the DataSource. Please add the unit wgsLegendCell and wgsTextSettings to the uses clause of your delphi main formular.

```
(48) ...
procedure TForm1.ShowLegend;
var
    LegendCell : TWGSLegendCell;
begin
    // First we add a new row to the legend ...
    WGSDataSource1.Legend.AddBlankRow(100,1);
    // ... then we create a new TWGSLegendCell
    LegendCell := TWGSLegendCell.Create;
    LegendCell.width := 100;
    LegendCell.Parent := WGSDataSource1.Legend.rows[0];
    LegendCell.textSettings := TWGSTextSettings.Create;
    LegendCell.textSettings.Parent := LegendCell;
    LegendCell.textSettings.color := clWhite;
    LegendCell.text := 'Hello World';
    // and add this cell to the row
    WGSDataSource1.Legend.rows[0].AddCell(LegendCell);
    // Finally we invoke the printing preview
    KatPrintPreview1.PrintOptions.PrintLegend := true;
    KatPrintPreview1.PrintOptions.PrintGraph := true;
    KatPrintPreview1.PrintOptions.PrintTable := true;

    KaTPrintPreview1.DoPrint(true,false);
end;
```

- After we have shown how to set up a basic legend including a row and a cell, we will show here how to add a border. A legend border is always defined by its start row, its start cell, its end row and its end cell. If you want to see the result of this code snippet you have to insert this code in the previous code example before the printing preview is shown.

```
(49) ...
procedure TForm1.AddBorder;
var
    LegendBorder : TWGSLegendBorder;
begin
    LegendBorder := TWGSLegendBorder.Create;
    LegendBorder.startRow := 0;
    LegendBorder.startCell := 0;
    LegendBorder.endRow := 0;
    LegendBorder.endCell := 0;
    LegendBorder.color := clRED;
    LegendBorder.thickness := 5;

    WGSDataSource1.Legend.addBorder(LegendBorder);
end;
```


Localization

Currently the Gantt Control Component is available for the both languages German and English. The following table shows all functions/methods defined in the unit `wgsLocalization`. Simply add the unit `wgsLocalization` to the uses clause of your main form and call the `CreateDefaultRessources_ENG` method to localize the Gantt Control Component to English.

Function	Description
<code>CreateDefaultRessources_GER</code>	Localizes the Gantt Control Component to the German language.
<code>CreateDefaultRessources_ENG</code>	Localizes the Gantt Control Component to the English language.
<code>SetRessourceString (Name:String;Value:String)</code>	Localizes a single language dependent string value and assigns a value to it.
<code>GetRessourceValue (Name:String):String;</code>	Returns the value for a given language dependent string.

Internally the both methods - `CreateDefaultRessources_ENG()` and `CreateDefaultRessources_GER()` calls the `SetRessourceString(Name:String; Value:String)` for all given language dependent resource string and defines their translation in this way.

- The following table summarizes all language dependent resource strings and their value for the English language.

```
(50) procedure CreateDefaultRessources_ENG;
begin
    SetRessourceString('autonumbercolumn','No');
    SetRessourceString('insertnewpertbarmsg1','Insert a new pert bar by dragging the mouse. ');
    SetRessourceString('insertnewpertbarmsg2','');
    SetRessourceString('pp_caption','Printing preview');
    SetRessourceString('pp_menu_document','Document');
    SetRessourceString('pp_menu_goto','Go to');
    SetRessourceString('pp_menu_print','Print...');
    SetRessourceString('pp_menu_pagesetup','Page setup...');
    SetRessourceString('pp_menu_close','Close');
    SetRessourceString('pp_menu_firstpage','First page');
    SetRessourceString('pp_menu_prevpage','Previous page');
    SetRessourceString('pp_menu_nextpage','Next page');
    SetRessourceString('pp_menu_lastpage','Last page');
    SetRessourceString('pp_hint_firstpage','First page');
    SetRessourceString('pp_hint_prevpage','Previous page');
    SetRessourceString('pp_hint_nextpage','Next page');
    SetRessourceString('pp_hint_lastpage','Last page');
    SetRessourceString('pp_hint_caption','Caption');
    SetRessourceString('pp_hint_background','Background');
    SetRessourceString('pp_hint_border','Border');
    SetRessourceString('pp_hint_footerheader','Page footer and header');
    SetRessourceString('pp_hint_chartsettings','Chart settings');
    SetRessourceString('pp_hint_print','Print...');
    SetRessourceString('pp_hint_pagesetup','Page setup...');
    SetRessourceString('pp_label_zoomfactor','Zoom factor:');
    SetRessourceString('pp_label_profile','Profile:');
    SetRessourceString('pp_hint_newprofile','save current settings as a new profile');
    SetRessourceString('pp_hint_saveprofile','save current profile');
    SetRessourceString('pp_hint_renameprofile','rename current profile');
    SetRessourceString('pp_hint_deleteprofile','delete profile');

    SetRessourceString('pp_mi_left','Left');
    SetRessourceString('pp_mi_right','Right');
    SetRessourceString('pp_mi_top','Top');
    SetRessourceString('pp_mi_center','Center');
    SetRessourceString('pp_mi_bottom','Bottom');
    SetRessourceString('pp_mi_header','Header');
    SetRessourceString('pp_mi_footer','Footer');
    SetRessourceString('pp_mi_days','day(s)');

    SetRessourceString('pp_mi_page','Page');
    SetRessourceString('pp_mi_paperformat','Page format');
    SetRessourceString('pp_sr_page','Page ');
    SetRessourceString('pp_pf_portrait','Portrait');
    SetRessourceString('pp_pf_landscape','Landscape');

    SetRessourceString('pp_ps_pagesetup','Page setup');
```



```

SetResourceString('pp_ps_printeroptions', 'Printer setup');
SetResourceString('pp_ps_caption', 'Caption');
SetResourceString('pp_ps_border', 'Border');
SetResourceString('pp_ps_background', 'Background');
SetResourceString('pp_ps_headerfooter', 'Page header and footer');
SetResourceString('pp_ps_ganttchart', 'Gantt chart options');

SetResourceString('pp_ps_printerpaperformat', 'Printer and paper format');
SetResourceString('pp_ps_preview', 'Preview');
SetResourceString('pp_ps_layout', 'Layout');
SetResourceString('pp_ps_scaling', 'Scaling');
SetResourceString('pp_ps_pageborder', 'Page borders');
SetResourceString('pp_ps_fittopage', 'Fit to one page');
SetResourceString('pp_ps_offsetleft', 'Offset left');
SetResourceString('pp_ps_offsettop', 'Offset top');

SetResourceString('pp_ps_preview', 'Preview');
SetResourceString('pp_ps_horizontalalignment', 'Horizontal alignment');
SetResourceString('pp_ps_verticalalignment', 'Vertical alignment');

SetResourceString('pp_bn_font', 'Font');
SetResourceString('pp_bn_background', 'Background');

SetResourceString('pp_ps_printcaption', 'Print caption');
SetResourceString('pp_ps_caption_eachpage', 'Each Page');
SetResourceString('pp_ps_caption_centered', 'Centered');

SetResourceString('pp_ps_border_printborder', 'Print border');
SetResourceString('pp_ps_border_borderwidth', 'Border width');
SetResourceString('pp_bn_border_changecolor', 'Color');

SetResourceString('pp_ps_background_useimage', 'Use image');
SetResourceString('pp_ps_background_position', 'Position');
SetResourceString('pp_bn_border_OpenImage', 'Open Image');

SetResourceString('pp_ps_border_item1', 'stretched');
SetResourceString('pp_ps_border_item2', 'centered');
SetResourceString('pp_ps_border_item3', 'tiled');

SetResourceString('pp_ps_headerleft', 'Header left');
SetResourceString('pp_ps_headercenter', 'Header center');
SetResourceString('pp_ps_headerright', 'Header right');
SetResourceString('pp_ps_footerleft', 'Footer left');
SetResourceString('pp_ps_footercenter', 'Footer center');
SetResourceString('pp_ps_footerright', 'Footer right');

SetResourceString('pp_ps_content', 'Content');
SetResourceString('pp_ps_image', 'Image');
SetResourceString('pp_ps_text', 'Text');
SetResourceString('pp_ps_autotextfields', 'Autotext fields');

SetResourceString('pp_ht_time', 'Time');
SetResourceString('pp_ht_date', 'Date');
SetResourceString('pp_ht_datetime', 'Time and Date');
SetResourceString('pp_ht_page', 'Page number');
SetResourceString('pp_ht_pagecount', 'Page count');
SetResourceString('pp_ht_pageof', 'Page # of #');

SetResourceString('pp_ps_timerange', 'Time range');
SetResourceString('pp_ps_selection', 'Selection');
SetResourceString('pp_ps_auto', 'All');
SetResourceString('pp_ps_timerange_start', 'Start');
SetResourceString('pp_ps_timerange_end', 'End');
SetResourceString('pp_ps_additionaldays', 'Show additional days');

SetResourceString('pp_ps_calendric_week', 'Every calendar week on a new page');
SetResourceString('pp_ps_timescale_under', 'Timescale under the gantt graph');
SetResourceString('pp_ps_coloneachpage', 'Print column on each page');

SetResourceString('pp_bc_barcaption', 'Bar caption');
SetResourceString('pp_bc_printbarcaption', 'Print bar caption');
SetResourceString('pp_ps_pagination', 'Pagination');
SetResourceString('pp_ps_printconnections', 'Print connections');

SetResourceString('pp_ps_nobreakonrow', 'No page break within a row');
SetResourceString('pp_ps_nobreakoncolumn', 'No page break within a column');

SetResourceString('pp_bn_Cancel', 'Cancel');
SetResourceString('pp_tl_printing', 'Printing...');
SetResourceString('pp_tl_print', 'Printer');
SetResourceString('pp_tl_printrange', 'Print range');
SetResourceString('pp_tl_options', 'Options');
SetResourceString('pp_tl_allpages', 'All pages');
SetResourceString('pp_tl_currentpage', 'Current page');
SetResourceString('pp_tl_pages', 'Pages:');
SetResourceString('pp_ps_Copies', 'Copies:');
SetResourceString('pp_ps_Monochrome', 'Monochrome');
SetResourceString('pp_ps_hide_background', 'Hide background');
SetResourceString('pp_pp_Prefix_printmsg', 'Print page');
SetResourceString('pp_of', 'of');

```

```

SetRessourceString('pp_op_savesettings','Save settings');
SetRessourceString('pp_op_usecurrentname','Use current profile');
SetRessourceString('pp_op_useanewname','Create a new profile');
SetRessourceString('pp_op_newprofile','Create a new profile');
SetRessourceString('pp_op_copyof','Copy of');

SetRessourceString('pp_msg_noprinter_pre','There is no access to the printer device: ');
SetRessourceString('pp_msg_noprinter_post','" Please check the system device status. ');
SetRessourceString('pp_msg_noprinter','No printer device detected!');
SetRessourceString('pp_msg_printrange','Illegal print range!');
SetRessourceString('pp_msg_deleteprofile_pre','Do you want to delete the profile : ');
SetRessourceString('pp_msg_deleteprofile_post','" ?');
SetRessourceString('pp_msg_rename','Rename');
SetRessourceString('pp_msg_newname','New name');
SetRessourceString('pp_msg_emptynewname','The profile name is empty!');
SetRessourceString('pp_msg_alreadyexists','The profile does already exists. ');

SetRessourceString('df_calendarweekstring','CW');
SetRessourceString('df_monthweekstring','W');
SetRessourceString('df_quarterlongstring','Quarter');
SetRessourceString('df_quartershortstring','Q');

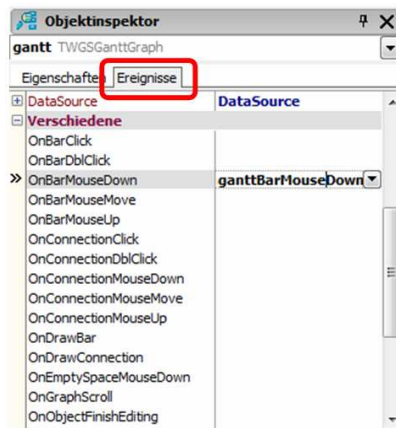
end;

```

You can overwrite any existing RessourceString with a new value.

Events

The Gantt Control Component defines several events that are accessible through Delphi. To access an event please select a **TWGSPertGraph** component, a **TWGSGanttGraph** component or the **TWGSDataSource** component in your Delphi application. After that, open the delphi object inspector or press [F11].



After you have opened the object inspector, please select the tab sheet “events”. Here you can see all the events the selected component provides to you.

To add some application code to an event, simply double-click on the event. Delphi will automatically create the correct empty code for the event handler.

The following tables show all events of the GanttGraph or PertGraph and a corresponding brief description.

WGSGanttGraph.OnClick(Sender: TObject) The OnClick event fires as soon as the user clicks on the GanttGraph, including all parts such as the time scale or the header cells.
WGSGanttGraph.OnDbClick(Sender: TObject) The OnDbClick event fires as soon as the user double-clicks on the GanttGraph, including all parts such as the time scales and the header cells etc.
WGSGanttGraph.OnKeyDown(Sender: TObject; var Key: Word; Shift: TShiftState); The OnKeyDown event fires as soon as the user presses down a key.
WGSGanttGraph.OnKeyPress(Sender: TObject; var Key: Char); The OnKeyPress event fires as soon as the user presses down a key.
WGSGanttGraph.OnKeyUp(Sender: TObject; var Key: Word; Shift: TShiftState); If the user releases a key the OnKeyUp event will fire.
WGSGanttGraph.OnMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer); As soon as a mouse button is clicked somewhere in the Gantt Graph – the OnMouseDown event will fire.
WGSGanttGraph.OnMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer); If the user moves the mouse over the GanttGraph or over parts of the GanttGraph the event OnMouseMove will be fired. The parameter <i>Shift</i> can be used to determine whether keys, such as [CTRL], [SHIFT] or [ALT] has been pressed while moving the mouse cursor. X and Y specifies the position of the mouse cursor.
WGSGanttGraph.OnGanttMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer); If the user releases a mouse button the event OnGanttMouseUp will be fired. The parameters <i>Button</i> (<i>mbLeft</i> , <i>mbRight</i> , <i>mbDouble</i>) specifies the mouse button that was released.
WGSGanttGraph.OnResize(Sender: TObject); If the GanttGraph component is forced to resize itself the event OnResize will be fired.
WGSGanttGraph.OnBarClick(Bar: TWGSGanttBar); If the user clicks on a bar (task bar, pert bar, progress bar, text bar, image bar or milestone bar) the OnBarClick event will be triggered. The parameter <i>Bar</i> specifies the bar that was clicked.

WGSGanttGraph.OnBarDbClick(Bar: TWGSGanttBar);

If the user double-clicks on a bar the event OnBarDbClick will be fired. The parameter `Bar` specifies the bar that was clicked.

WGSGanttGraph.OnGanttBarMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

As soon as a mouse button is clicked while the mouse cursor is over a gantt bar the OnGanttBarMouseDown event is triggered. The parameter `Button` (mbLeft, mbRight, mbDouble) specifies the Button that has been pressed.

WGSGanttGraph.OnGanttBarMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);

If the user moves the cursor over a bar object such as task bar, pert bar, progress bar, milestone bar, text bar or image bar the OnGanttBarMouseMove event will be fired. The parameter `Sender` specifies the bar that is under the mouse cursor. To access properties of the bar, it is recommended to check first whether the `Sender` object is from a specific bar type. After that you can typecast the `Sender` object. How this can be done is shown in the following code example:

- The principle of this code pattern can be applied to a lot of situation where an unspecified parameter such as `Sender: TObject` holds a derived object such as a GanttBar. In the code example below we first check whether `Sender` is from type `TWGSTaskBar` after that we can typecast the `Sender`.

```
(51) ...
procedure TForm1.WGSGanttGraph1BarMouseMove(Sender: TObject; Shift: TShiftState;
  X, Y: Integer);
begin
  if Sender is TWGSTaskBar then
    begin
      TWGSTaskBar(Sender).bufferTimeRight := 2;
    end;
end;
```

WGSGanttGraph.OnGanttBarMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

If the user releases a mouse button while the cursor is over a gantt bar the OnGanttBarMouseUp event will be fired. The parameter `Sender` specifies the bar that is under the mouse cursor while the mouse button is released. The parameter `Button` (mbLeft, mbRight, mbDouble) determines the mouse button that was released.

WGSGanttGraph.OnConnectionClick(Connection: TWGSBarConnection);

If the user clicks on a connection between two gantt bars or two pert bars the event OnConnectionClick will be fired. The parameter `Connection` holds the connection that was clicked.

WGSGanttGraph.OnConnectionDbClick(Connection: TWGSBarConnection);

If the user double-clicks on a connection between two gantt bars or two pert bars the event OnConnectionDbClick will be fired. The parameter `Connection` holds the connection that was double-clicked.

WGSGanttGraph.OnConnectionMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

If the user presses a mouse button while the mouse cursor is over a connection between two gantt bars or two pert bars the event OnConnectionMouseDown will be fired. `Sender` holds the BarConnection object.

WGSGanttGraph.OnConnectionMouseMove(Sender: TObject; Shift: TShiftState; X, Y: Integer);

If the mouse cursor moves over a connection between two bars the OnConnectionMouseMove event will be raised. The parameter `Shift` can be used to determine whether keys, such as [CTRL], [SHIFT] or [ALT] has been pressed while moving the mouse cursor.

WGSGanttGraph.OnConnectionMouseUp(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

If the mouse button is released while the mouse cursor is over a connection the OnConnectionMouseUp event will be triggered.

WGSGanttGraph.OnEmptySpaceMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

If the user clicks on the gantt graph and he does **not** click a bar or a cell of the tree, the OnEmptySpaceMouseDown event will be triggered. So every time the user clicks the ganttgraph and under the mouse cursor is no editable or selectable object the OnEmptySpaceMouseDown event will be fired. The values for the x and y coordinates specifies the position where the user has clicked.

WGSGanttGraph.OnGraphScroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);

If the GanttGraph is scrolled horizontally the OnGraphScroll event is raised. `ScrollCode` describes how the scrollbar is scrolled (scLineUp, scLineDown, scPageUp, scPageDown, scPosition, scTrack, scTop, scBottom, scEndScroll). `ScrollPos` specifies the position of the Scroll-Button within the scrollbar.

WGSGanttGraph.OnObjectFinishEditing(Sender: TObject);

If the user finishes the editing of an object the event OnObjectFinishEditing will be raised. Please note whenever the user finished editing a cell of the treegrid the event OnObjectFinishEditing will be triggered – the parameter Sender contains the current edited object (for example a tree cell). The following code shows how to access the cell.

- The following code example shows how to access a single cell using the OnObjectFinishEditing event. For this example please add an empty simple text cell column to the tree grid. After finish editing a text cell of this column we will overwrite the edited cell value within the tree grid.

```
(52) uses ... ,
      wgsTreeCell,
      wgsTreeSimpleTextCell;

...
procedure TForm1.OnObjectFinishEditing(Sender: TObject);
var
  Row, Col : integer;
begin
  if (Sender is TWGSTreeCell) then
    begin
      // tree cell
      Col := TWGSTreeCell(Sender).ColumnIndex;
      Row := TWGSTreeRow(TWGSTreeCell(Sender).Parent).RowIndex;
      if Col=1 then // only if the second column has been edited
        WGSGanttGraph1.tree.rows[Row].cells[Col].Text := 'CELL EDITED';
      end;
    end;
end;
```

WGSGanttGraph.OnObjectStartEditing(Sender: TObject);

As described above the OnObjectStartEditing event is the counterpart of the OnObjectFinishEditing event. The OnObjectStartEditing event is fired each time the user start editing an editable object such as tree cells. Please have a look on the previous code example to know how to access a single cell of the tree grid.

WGSGanttGraph.OnRowFocus(Sender: TObject);

Each time the user focuses another row the OnRowFocus event is fired. If you select another row the parameter Sender that is a TWGSTreeRow specifies the previously focused row.

WGSGanttGraph.OnSelectedRowsChanged(SelectedRows: TObjectList);

Each time the user selects a row the OnSelectedRowsChanged will be triggered.

WGSGanttGraph.OnSplitterMove(Sender: TObject);

There is a splitter between the gantt table and the tree grid. After the splitter has been dragged to a new position with the mouse or the splitter has been double-clicked in order to snap to a new position the OnSplitterMove event has been fired.

WGSGanttGraph.OnTreeCellMouseDown(Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer);

If the user clicks a mouse button while the cursor is over a treegrid cell, the OnTreeCellMouseDown event will be raised. The parameter Sender specifies the TreeCell that has been clicked, the parameter Button determines the button type (mbLeft, mbRight, mbDouble) that was clicked.

WGSGanttGraph.OnTreeScroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);

If the tree grid is scrolled horizontally the OnGraphScroll event is raised. ScrollCode describes how the scrollbar is scrolled (scLineUp, scLineDown, scPageUp, scPageDown, scPosition, scTrack, scTop, scBottom, scEndScroll). ScrollPos specifies the position of the Scroll-Button within the scrollbar.

WGSGanttGraph.OnVerticalScroll(Sender: TObject; ScrollCode: TScrollCode; var ScrollPos: Integer);

If the GanttGraph (tree grid and gantt table) is scrolled vertically the OnVerticalScroll event is raised. ScrollCode describes how the scrollbar is scrolled (scLineUp, scLineDown, scPageUp, scPageDown, scPosition, scTrack, scTop, scBottom, scEndScroll). ScrollPos specifies the position of the Scroll-Button within the scrollbar.

The following tables shows all events of the DataSource and a brief description

WGSDatasource.OnAfterBarAdd(Bar: TWGSGanttBar);

After a bar has been added to the DataSource the OnAfterBarAdd event will be raised. The parameter Bar specifies the bar that has been added.

WGSDatasource.OnAfterBarDelete(Bar: TWGSGanttBar);

After a bar has been deleted from the DataSource the OnAfterBarDelete event will be raised. The parameter Bar

specifies the bar that has been deleted.

WGSDDataSource.OnAfterBarMove(Bar: TWGSGanttBar);

After a bar has been moved the OnAfterBarMove event will be triggered. The parameter `Bar` specifies the Bar that has been moved.

WGSDDataSource.OnAfterBarResize(Bar: TWGSGanttBar);

After a bar has been resized the OnAfterBarResize event will be triggered. The parameter `Bar` specifies the Bar that has been resized.

WGSDDataSource.OnAfterConnectionAdd(Connection: TWGSBarConnection);

After a connection has been established between two bars the OnAfterConnectionAdd event will be raised. The parameter `Connection` specifies the current connection that will be added to the data source.

WGSDDataSource.OnAfterConnectionDelete(Connection: TWGSBarConnection);

If a connection has been deleted the OnAfterConnectionDelete event is fired. The parameter `Connection` specifies the deleted connection.

WGSDDataSource.OnBarResize(Bar: TWGSGanttBar; changeX: Real; changeY: Integer; Right: Boolean; var Allowed: Boolean);

After a bar has been resized the OnBarResize event will be fired. The parameter `Bar` specifies the Bar that has been resized. The parameter `changeX` specifies the amount of time units the bar has changed. For example if the `TimeMode` was set to `tmDay` and the bar has been downsized by one day, the value for `changeX` is -1. The parameter `Right` is true if the right side of a bar has been moved. If the left side of a bar has been moved the parameter `Right` is false. Within your own event handling code you can specify whether the resizing operation of the bar was valid. If so set `Allowed` to true otherwise set `Allowed` to false. If `Allowed` has been set to false the bar will not resize.

WGSDDataSource.OnBeforeBarAdd(Bar: TWGSGanttBar; var Cancel: Boolean);

If a new bar is added to the data source the event OnBeforeBarAdd will be triggered before the bar will be added to the datasource. If you set the parameter `Cancel` to true the bar you originally intended to add, will not be added to the datasource.

WGSDDataSource.OnBeforeBarDelete(Bar: TWGSGanttBar; var Cancel: Boolean);

Before a bar will be deleted the OnBeforeBarDelete event will be raised. Set `cancel` to true to abort the deletion of the bar.

WGSDDataSource.OnBeforeBarMove(Bar: TWGSGanttBar; var Cancel: Boolean);

Before a bar will be moved the OnBeforeBarMove event will be fired. Set the parameter `cancel` to true to abort the move operation of the bar.

WGSDDataSource.OnBeforeBarResize(Bar: TWGSGanttBar; var Cancel: Boolean);

Before a bar will be resized the OnBeforeBarResize event will be fired. Set the parameter `cancel` to true to abort the resize operation of the bar.

WGSDDataSource.OnBeforeConnectionAdd(Connection: TWGSBarConnection; var Cancel: Boolean);

Before a connection is added to the datadource the OnBeforeConnectionAdd event will be triggered. The `cancel` parameter specifies whether the operation should be performed.

WGSDDataSource.OnBeforeConnectionDelete(Connection: TWGSBarConnection; var Cancel: Boolean);

Before a connection will be deleted. The OnBeforeConnectionDelete event will be fired. Use the parameter `cancel` to abort the delete operation of the connection.

WGSDDataSource.OnChange(Operation: TWGSOperation);

Whenever there are changes made to the data source, the OnChange event will be fired. This includes editing, adding, deleting and moving objects. To determine the type of operation that will be performed use the parameter `operation`.

WGSDDataSource.OnDrawCell(rowIndex, cellIndex: Integer; cellRect: TRect; Cell: TWGSTreeCell; var vc: TWGSVisualContext; var UserDraw: Boolean);

If you want to change the visualization of a tree grid cell you can use the OnDrawCell event. This event will be fired each time a cell is drawn. The parameters `Cell`, `rowIndex` and `cellIndex` determine the `cell` that is currently being drawn. `CellRect` specifies the visible area of the cell. If you intend to handle the drawing of the cell and its content on your own, you have to set true for the parameter `UserDraw`. The visual context object `vc` forms an interface that holds several drawing operation that may be used to draw the cell and its content.

WGSDDataSource.OnGanttBarMove(Bar: TWGSGanttBar; NewRow: Integer; NewStartDate: TDateTime; var Allowed: Boolean);

After a gantt bar has been moved the OnGanttBarMove operation will be fired. The parameter `Bar` specifies the bar that has been moved. The parameter `NewRow` specifies the index of the row the gantt bar has been dragged to. `NewStartDate` specifies the new start date of the gantt row. The parameter `Allowed` should be set to `true` if the operation should be performed or `false` if the operation should be aborted.

WGSDDataSource.OnRedo(Operation: TWGSOOperation);

Whenever an operation is redone the OnRedo event will be fired, providing access to the undone operation using the parameter `Operation`.

WGSDDataSource.OnRowCollapse(SummaryBar: TWGSSummaryBar);

As soon as a row is collapsed the OnRowCollapse event will be fired.

WGSDDataSource.OnRowDelete(Row: TWGSGanttRow);

If a row will be deleted the OnRowDelete event will be raised. The parameter `Row` provides access to the row that should be deleted.

WGSDDataSource.OnRowExpand(SummaryBar: TWGSSummaryBar);

As soon as a row is expanded the OnRowExpand event will be fired.

WGSDDataSource.OnRowInsert(Row: TWGSGanttRow);

After a new row has been inserted to the gantt graph or the pert graph the OnRowInsert event will be fired. The parameter `Row` specifies the row that has been inserted.

WGSDDataSource.OnTreeCellButtonClicked(Sender: TWGSGanttObject);

If the tree grid contains a `TreeButtonEditCell` column the OnTreeCellButtonClicked event will raise if the user clicks on a tree cell button.

WGSDDataSource.OnTreeColumnMove(column: TWGSTreeHeaderCell; width: Integer);

If a column of the tree grid is moved the OnTreeColumnMove operation will be fired.

WGSDDataSource.OnTreeColumnResize(column: TWGSTreeHeaderCell; index: Integer);

If a column of the tree grid is redized the OnTreeColumnMove operation will be fired.

WGSDDataSource.OnUndo(Operation: TWGSOOperation);

Whenever an operation is undone the OnUndo event will be fired, providing access to the undone operation using the parameter `Operation`.

Time Format Value Table

As shown in the chapter *Time scales*, each time does have the property `TimeFormat` that specifies the caption of date cells within the time scale. Please note that the time mode and the time format of the timescale must fit to each other.

The following table shows all time formats, the corresponding time mode and a brief example.

TimeFormat	TimeMode	Example
tmfNone	(all)	
tmfHourStandard	tmHour	0, 1, 2, 3, ... , 23
tmfHourAMPM	tmHour	1 AM, 2 AM , 3 AM , ..., 11 PM
tmfHourTime	tmHour	00:00, 01:00, 02:00, ..., 23:00
tmfDayFull	tmDay	Monday, Tuesday, Wednesday, ...
tmfDayShort	tmDay	M, T, W, T, F, S, S
tmfDayOfYear	tmDay	1, 2, 3, ..., 364, 365
tmfDayOfMonth	tmDay	1, 2, 3, ..., 30, 31
tmfDayOfWeek	tmDay	1, 2, 3, 4, 5, 6, 7
tmfDayShortDayNumber	tmDay	Mo 5, Tu 6, We 7, Th 8, Fr 9, ...
tmfWeekYear	tmWeek	1.KW, 2.KW, 3.KW , ...52. KW
tmfWeekYearNr	tmWeek	1, 2, 3, 4, 5, ..., 52
tmfWeekMonth	tmWeek	1.W, 2.W, 3.W, 4.W
tmfWeekMonthNr	tmWeek	1, 2, 3, 4
tmfWeekStartDay	tmWeek	02/04/2007, 09/04/2007
tmfMonthFull	tmMonth	January, February, ..., December
tmfMonthMedium	tmMonth	Jan, Feb, ..., Dec
tmfMonthShort	tmMonth	J, F, M, ..., D
tmfMonthFullYear	tmMonth	January 2008, February 2008, December 2008
tmfMonthMediumYear	tmMonth	Jan 2008, Feb 2008, Dec 2008
tmfMonthShortYear	tmMonth	J2008, F2008, ..., D2008
tmfMonthNumber	tmMonth	1, 2, 3, ..., 12
tmfMonthNumberYear	tmMonth	1 2008, 2 2008, 3 2008, ..., 12 2008
tmfQuarterFull	tmQuarter	1. Quartal, 2. Quartal, 3.Quartal
tmfQuarterShort	tmQuarter	1.Q, 2.Q, 3.Q, 4.Q
tmfQuarterFullYear	tmQuarter	1. Quartal 2008, 2. Quartal 2008, 3. Quartal 2008
tmfQuarterShortYear	tmQuarter	1. Q 2008, 2.Q 2008, 3. Q 2008
tmfQuarterRoman	tmQuarter	I, II, III, IV
tmfYearFull	tmYear	2006, 2007, 2008, 2009
tmfYearShort	tmYear	06, 07, 08, 09
tmfDecade	tmDecade	2010- 2020