VCL

TMS TPlannerExport
DEVELOPERS GUIDE

Jul, 2018
Copyright © 1999 – 2018 by tmssoftware.com bvba
Web: http://www.tmssoftware.com
Email : info@tmssoftware.com

Table of contents

Using TPlannerExport to export to Excel files

The TPlannerExport component is a component that is an interface between the T(DB)Planner component and the Flexcel component suite to create Excel sheets from the information visualized in the T(DB)Planner. If you have not installed the Flexcel component suite, it can be obtained from http://www.tmssoftware.com/site/flexcel.asp

The TPlannerExport component encapsulates all code required to use Flexcel to create rich Excel files representing as close as possible what is visually displayed in the T(DB)Planner and offers a great way to share schedules with other people who might not have access to the application using the T(DB)Planner component.

Installing

When you have installed both T(DB)Planner and the Flexcel component suite, add the file PlannerExportReg.pas to your T(DB)Planner package file. Delphi or C++Builder should be smart enough to add a reference to the Flexcel package in the requires list. If not, add a reference to this package file as well. After compiling, the TPlannerExport should appear on the component palette in the TMS Planners tab.

Basic use

Drop TPlannerExport on the form, set its Planner property to the T(DB)Planner on the form and call following code to generate the XLS file:

```
plannerexport1.BeginExport;
plannerexport1.Export('schedule',1,1,false);
plannerExport1.Save('myplanner.xls', ExportType_Xls);
plannerExport1.EndExport;
```

Alternatively, the TPlannerExport could also be created & used fully runtime like the code:

```
var
  Wpe:  TPlannerExport;
begin
  Wpe := TPlannerExport.Create(nil);
  Wpe.AllowOverwritingFiles := true;
  try
    Wpe.Planner := Planner1;
    Wpe.BeginExport;
    try
      Wpe.Export('MySchedule', 1, 1, true);
      try
        Wpe.Save('Planner.xls', ExportType_Xls);
        ShellExecute(0, 'open','Planner.xls','','',SW_SHOWNORMAL);
      finally
        DeleteFile('Planner.xls');
      end;
    finally
      Wpe.EndExport;
    end;
  finally
    Wpe.Free;
  end;
```

```
      end;
```

Controlling how export is done

There are a few properties that you can set on the TPlannerExport to change its behaviour. This is a brief summary of how you can use them.

First of all, you can set where on the generated file the planner will go. This is set by the parameters on the line: exporter.Export("2005 Plan", 1, 1, true);

Here we are exporting to cell (1,1) (A1) but we could send the exported planner to any cell on the spreadsheet. With the last parameter, you can also control if you want to insert the planner into a spreadsheet (moving the rest of the sheet down) or just replace the existing cells.

You could export two different planners to the same sheet, by calling this line more than once:

```
exporter.Export("2005 Plan", 1, 1, true);
exporter2.Export("2005 Plan", 1, 1, true);
```

Important Note: When you export a planner to excel, columns are created and resized to fit the events. For example, a planner column with three overlapping events would have three corresponding columns on Excel, while the next planner column might have only one corresponding column on Excel.

If you try to put two planners one above the other, exporting the second planner will probably break the first one, as columns will be resized again. One way to avoid this is to set the MinimumColumnCount property of the TPlannerExport to the maximum expected overlapping events. On a more general case, the best solution is probably to export the planners to different worksheets:

```
exporter.Export("sheet1", 1, 1, true);
exporter2.Export("sheet2", 1, 1, true);
```

Other properties you might want to look for are ZoomX and ZoomY. On normal use, TPlannerExport tries to match the pixels on the planner to the pixels on Excel. But sometimes this is no good enough, because not the browser nor Excel use real pixels as their basic units, and the result might depend of the resolution, big fonts, etc. Or you just might want to make the exported planner smaller/larger than the original. By changing these two properties you can easily achieve this. And, since we are speaking of xls files, you can also fit the planner on one page for example, as it is discussed on the next section.

Customizing the exported file

As it has been mentioned before, we try not to cut corners on what you can do on the exported file, so the original FlexCel objects used by the TPlannerExport are available for you to do further customization.

There are actually two places where you can customize the output:

1) Before exporting.

One of the easiest ways to customize the export is not to start from a blank file. If you use:

```
exporter.BeginExport("template.xls");
```

instead of

```
exporter.BeginExport();
```

TPlannerExport will draw the planner over an existing template. You could have the company logo on that template, define headers and footers, or do a lot of work that might have to be done manually by code.

One often overlooked big advantage of using a starting template is that your users might be able to modify it without modifying the code. If for example the company logo changes, the only thing you need to do is to edit the template and change it, no need to touch the code.

Important Note: When you export, the columns and rows on the template will be resized to match the needed sizes for the planner. This might alter the look of your template, so it is a good idea to place things on column A, and export the planner to column B.

2) After exporting the xls file.

You can use the XlsFile object published by TPlannerExport to further modify the template once it has been generated.

For example, the following code would customize the exported xls file so it prints on one sheet wide by one sheet tall:

```
var
  Wpe: TPlannerExport;
begin
  Wpe := TPlannerExport.Create;
  Wpe.AllowOverwritingFiles := true;
  Wpe.Planner := Planner1;
  Wpe.BeginExport;
  Wpe.Export("2005 Plan", 1, 1, true);
  Wpe.Xls.PrintNumberOfHorizontalPages := 1;
  Wpe.Xls.PrintNumberOfVerticalPages := 1;
  Wpe.Xls.PrintToFit := true;
  Wpe.EndExport();
end;
```

Note that this is equivalent to using a template (as in method 1) where the template has been configured on the Page->Setup dialog from Excel.

You have the full XlsFile object exposed here so you can do virtually anything with the file, insert cells, modify values, delete sheets, etc. The XlsFile object is documented on FlexCel documentation.