

# Embedding Spreadsheet Functionality Using the TMS TAdvSpreadGrid

by Craig Murphy

By the time you read this article, I will have been working with my current employer for four years. During that time, I have been involved in the development of many “cost management” applications. Asset management/valuation and cost estimating applications appear to be a common requirement and are applications I have spent a considerable amount of time designing, developing, testing, etc. My user-base consists of cost-estimators, data input clerks, quantity surveyors – i.e. people who are generally comfortable using office products such as Microsoft Excel. In fact, I’m happy to admit that I have the benefit of working with users who are able to exploit the true power of Microsoft Excel. As such, some of my applications have a workbook look and feel, just like Microsoft Excel.

However, there is one thing that a majority of my users say: “I can do a lot of what your application does using Microsoft Excel”. That’s probably true; however I often have to explain the rationale behind database theory, Client/Server, multi-user update, etc. Usually the pre-cursor to our applications is a Microsoft Excel workbook that has outgrown the capabilities of Excel. Often the original “author” of the workbook is able to verbally/graphically explain *what* they need the system to do – but they don’t know how to develop the solution. Convincing a user to let go of a spreadsheet-based solution can be difficult, so frequently we re-use the existing office technologies, such as Microsoft Excel, Microsoft Word and even Microsoft Project. The Component Object Model (COM) has made this possible with the majority of Microsoft products.

However, what if the application is being deployed in an environment without Microsoft Excel? This isn’t an uncommon scenario – there are still a lot of our clients using Lotus products, if not for their office solutions, then for their groupware. Similarly, Microsoft Excel has licensing requirements – and since there are costs involved, using Excel might be prohibitive.

Since users don’t need to know the full picture in order to use an application, building an application that relies on Microsoft Excel might be overkill. Similarly, as developers, it may not be cost-effective for us to learn how to interface/integrate with other office products from other vendors. We could use some of Delphi’s components, such as TF1Book – the Formula One spreadsheet component. However, since it lives on the ActiveX tab, you can be sure that there is an OCX that may need to be deployed at your client site. Another solution is to seek out a fully blown third party VCL component that provides us with spreadsheet functionality. One such component is the TMS TAdvSpreadGrid.

## TAdvSpreadGrid Background

TMS Software originally produced an advanced string grid, TAdvStringGrid. Naturally this component offered everything you would expect from Delphi’s TStringGrid, and whole lot more. As time went by, TMS Software developed TAdvSpreadGrid, a spreadsheet-like component based on TAdvStringGrid. As you might expect,

TAdvSpreadGrid offers a number of built-in functions. Mathematical functions are in abundance: there are functions for calculating absolute values (ABS), cosines (COS, ACOS), logarithms (LOG2, LOG10) – too many to mention here.

Cell range functions include the traditional SUM, AVERAGE, MIN, MAX, COUNT, and the usual functions that deal with deviations and variances. Date & Time functions are also catered for – functions such as WEEKDAY, TODAY, NOW, DAY, MONTH, YEAR, HOUR, MINUTE, etc. allow us to perform date/time arithmetic in the cells of a TAdvSpreadGrid. Decision support functions, such as LT (less than), GT (greater than), EQ (equal) and CHOOSE allow us to create complicated conditional statements. A number of “constants” are built-in too, namely: PI, E, True, False.

TAdvSpreadGrid now enjoys support from the ESBMaths library – this provides TAdvSpreadGrid with a collection of constants and functions that augment the existing built-in constants and functions. These functions are useful, if a touch obscure, but they do have a place in a spreadsheet. For example, one such function offered by the ESBMaths is DISTANCE (X1, Y1, X2, Y2) – this returns the straight-line distance between (X1, Y1) and (X2, Y2).

## TAdvSpreadGrid in Action

Whilst I have a few reasonably complicated examples of the TAdvSpreadGrid component in use, the best way to convey its ease of use is by means of a simple example. The example, as depicted by Figure 1, demonstrates a number of the features in action. Firstly, we see how cells can be added together using the SUM function. Then we take a look at a few of the time and date functions. As you would expect, TAdvSpreadGrid is heavily event-driven. Thus it is possible to customise the appearance of the spreadsheet by providing our own code for such events as OnDrawCell and OnGetCellColor – Listing 2 demonstrates how we colour rows 4 and 9 black.



Figure 1

```

with asgDemo Do begin
    FixedCols:=0;
    ColCount := 2;
    RowCount := 11;

    Cells[0,1] := 'A';
    Cells[0,2] := 'B';
    Cells[0,3] := 'A+B=';

    Cells[1,1] := '21';
    Cells[1,2] := '21';

    // Allocate names to the cells...
    CellName[1,1] := 'A';
    CellName[1,2] := 'B';

    // ...then use the cell names
    // instead
    // of the absolute cell references
    Cells[1,3] := '=SUM(A,B)';
    // OR Cells[1,3] :=
    //     '=SUM(R1C2,R2C2)';

    Cells[0,5] := 'WEEKDAY';
    Cells[0,6] := 'MONTH';
    Cells[0,7] := 'YEAR';
    Cells[0,8] := 'YEAR+1';

    Cells[1,5] := '=WEEKDAY(TODAY)';
    Cells[1,6] := '=MONTH(TODAY)';
    Cells[1,7] := '=YEAR(TODAY)';
    Cells[1,8] := '=YEAR(TODAY)+1';

    ColWidths[1]:=140;

    Cells[1,10] := '<IMG src="file:///
    +
    ExtractFilePath(Application.ExeName)
    + 'dmag.gif">';
    RowHeights[10]:=50;
end;

```

**Listing 1 – Setting up a TAdvSpreadGrid programmatically**

```

procedure
    TForm1.asgDemoGetCellColor(Sender:
    TObject; ARow, ACol:
    Integer;
    AState: TGridDrawState; ABrush:
    TBrush; AFont: TFont);
begin
    if (ARow in [4,9]) then begin
        ABrush.Color := clBlack;
    end;
end;

```

**Listing 2 – Colouring rows black**

## Named Cells

One key feature that is available as part of the TAdvSpreadGrid is the notion of named cells. Obviously it is very easy to build a spreadsheet with absolute cell references, e.g. R1C1. As user requirements are rarely absolute, I have found it safer to give cells names. This allows me to reference a particular cell by a meaningful name rather than by row and column. It also allows the spreadsheet to be a little more flexible. After all, I am not

interested in fixed cell references, so the actual cell that contains the value of interest could move about the spreadsheet without affecting the rest of my code. Whilst on the subject of cell references, TAdvSpreadGrid offers properties that allow the cell naming to be configured. The cell name can be either in RxCy format or in A1 format.

## HTML with everything

The TMS TAdvStringGrid component (TAdvSpreadGrid's parent), allows HTML tags to be used to format cells. Naturally, this feature is also available in the downstream components, including TAdvSpreadGrid. TMS Software offers a subset of the HTML specification for use with the vast majority of their components – this subset is known as MiniHTML. MiniHTML allows us to use HTML tags to format the contents of cells – this is how the developers' magazine logo appears in Figure 1: via an <IMG> tag.

The inclusion of HTML support coupled with the ability to control the rendering of each and every cell, gives us the opportunity to create sophisticated data collection forms. In particular, it is possible to include common controls (check boxes, radio buttons, drop-downs, etc.) in a grid's cell; this adds to the power of the TAdvSpreadGrid and TAdvStringGrid components. The ability to re-create existing paper-based forms, and improve upon them, is an exceptionally useful feature.

## XML is everywhere

You didn't think I could write an article without mentioning XML? The majority of the TMS grids support a variety of export formats. Apart from the obvious Comma Separated Value (CSV) format (which lends itself to saving spreadsheet data), the modern equivalent, XML, is also supported. Saving a TAdvSpreadGrid to an XML format is perhaps a little strange, so I will demonstrate how to save the TAdvColumnGrid to an XML format. Unlike CSV files, we need to provide information about the XML that is to be generated. Notably, we must provide the root element (in this case AIRCOOLER), the record elements (in this case ITEM) and a string list that defines the column names that are to be used (in this case WEIGHT and HOURS). Listing 3 presents the code that is required to save grid data to an XML file. Listing 4 presents the XML file that Listing 3 creates.

```

procedure TForm1.btnSaveClick(Sender:
    TObject);
var
    sl:TStringList;
begin
    sl:=TStringList.Create;
    sl.Add('WEIGHT');
    sl.Add('HOURS');

    acgLookup.SaveToXML(
        'AIRCOOLER.XML',
        'AIRCOOLER',
        'ITEM',sl);

    sl.Free;
end;

```

**Listing 3 – Saving grid data as XML**

```

<?xml version="1.0"?>
<AIRCOOLER>
<ITEM>
<WEIGHT>Weight</WEIGHT>
<HOURS>Hours</HOURS>
</ITEM>
<ITEM>
<WEIGHT>10</WEIGHT>
<HOURS>164</HOURS>
</ITEM>
<ITEM>
<WEIGHT>20</WEIGHT>
<HOURS>328</HOURS>
</ITEM>
<ITEM>
<WEIGHT>30</WEIGHT>
<HOURS>492</HOURS>
</ITEM>
<ITEM>
<WEIGHT>40</WEIGHT>
<HOURS>656</HOURS>
</ITEM>
</AIRCOOLER>

```

**Listing 4 – Grid data as XML**

Looking back at Figure 1, you'll see there is a Show Formula checkbox. Figure 3 presents the spreadsheet after the Show Formula checkbox has been selected. As you can see, rather than displaying the cell values the formula is shown instead. This is particularly useful if the spreadsheet has been created on the fly; it is often very useful whilst debugging arithmetic errors.



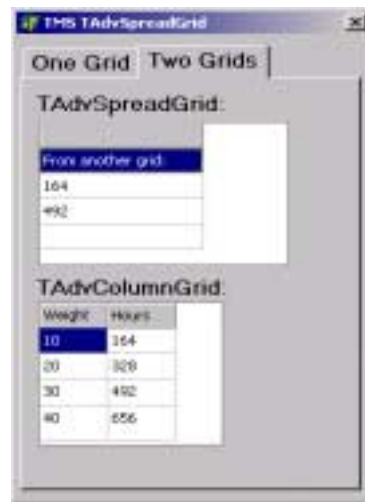
**Figure 2**

## Working with multiple grids

TMS Software produce a number of grid components – as I've already mentioned, it all started with the TAdvStringGrid. In addition to TAdvSpreadGrid there is a TAdvColumnGrid, and a data-aware version of the TAdvStringGrid component. One of the useful features found in Microsoft Excel is the ability to have more than one worksheet within a workbook. Typically, I find my users hide some worksheets that perform a lot of calculations. They then reference the hidden worksheet from another worksheet, extracting and displaying only the information they are interested in. The TMS grid components allow us to emulate such behaviour via the use of a grid binder. As you might expect, if you have a number of grids on, say a TPageControl, the TGridBinder component allows us to group together a number of grids at design time. The beauty

of this approach is realised by the fact it will allow us to reference cells in other grids, i.e. a worksheet can reference cells in another worksheet. It gets better: the binder component allows us to mix and match between grids, i.e. the grids in the binder do not need to be of the same type.

Figure 3 demonstrates more than one grid in action. There is a spreadsheet and another TMS grid, the TAdvColumnGrid – both are logically grouped together using a TGridBinder component. Listing 3 demonstrates how we can populate the two grids with some information. I have highlighted the important lines in bold – these are the lines that allow cross-grid referencing.



**Figure 3**

```

with acgLookup do begin
  Cells[0,1] := '10';
  Cells[0,2] := '20';
  Cells[0,3] := '30';
  Cells[0,4] := '40';

  Cells[1,1] := '164';
  Cells[1,2] := '328';
  Cells[1,3] := '492';
  Cells[1,4] := '656';
end;

with asgDemo2 Do begin
  FixedCols := 0;
  ColCount := 1;

  Cells[0,1] := 'From another
  grid: ';
  Cells[0,2] :=
    GridBinder.Grids.GridByName
    ['acgLookup'].Cells[1,1];
  Cells[0,3] :=
    GridBinder.Grids.GridByName
    ['acgLookup'].Cells[1,3];

  ColWidths[0] :=
    Canvas.TextWidth(UPPERCASE
    ('From another grid:'));
end;

```

**Listing 3 – Setting up multiple grids programmatically**

## Component Commotion

Time is central to the use of components. We can save time when we use a component, but we can also spend (waste) a lot of time looking for components, installing them, deciding that they're not useful enough, and uninstalling them. In fact, I spent a lot of time during the early Delphi years (1995, 1996) looking at component after component. I spent so much time then, that today I hardly spend any time at all looking at new components.

I rely on word-of-mouth and recommendations before I will install a new component. For example, with the exception of InfoPower, TMS, and OpenXML, my Delphi 6 component palette has two components (one of which I wrote myself). Hence I put together this short article about one of the components that I find increasingly useful – hopefully it is enough to allow you make the decision about the usefulness of the TAdvSpreadGrid component.



## And finally...

I hope this brief look at one of the TMS components has whetted your appetite. If it has, a visit to the TMS web site is well worth it. The majority of the components are available outside of the component pack, i.e. you can select those individual components that interest you most. I have found the support offered by TMS Software to be excellent; e-mails are responded to very quickly – often within sixty minutes of receipt.

## Useful URLs

- TMS Software: <http://www.tmssoftware.com>
- ESBSMaths library: <http://www.esbconsult.com.au>

*Craig works as an Enterprise Developer (and Dilbert Evangelist!) for Currie & Brown (<http://www.currieb.com>) – their primary business is quantity surveying, cost management and project management. He can be reached via e-mail at: [Craig.Murphy@currieb.co.uk](mailto:Craig.Murphy@currieb.co.uk) or [Craig@isleofjura.demon.co.uk](mailto:Craig@isleofjura.demon.co.uk)*



From June 15, 1999 Defence Science and Technology Organization LectureSeries, Melbourne, and staff reports. The reuse of some object-oriented code has caused tactical headaches for Australia's armed forces. As virtual reality simulators assume larger roles in helicopter combat training, programmers have gone to great lengths to increase the realism of their scenarios, including detailed landscapes and, in the case of the Northern Territory's Operation Phoenix, herds of kangaroos (since disturbed animals might well give away a helicopter's position).

The head of the Defence Science & Technology Organization's Land Operations/Simulation division reportedly instructed developers to model the local marsupials' movements and reactions to helicopters.

Being efficient programmers, they just re-appropriated some code originally used to model infantry detachment reactions under the same stimuli, changed the mapped icon from a soldier to a kangaroo, and increased the figures' speed of movement.

Eager to demonstrate their flying skills for some visiting American pilots, the hotshot Aussies "buzzed" the virtual kangaroos in low flight during a simulation. The kangaroos scattered, as predicted, and the visiting Americans nodded appreciatively... then did a double-take as the kangaroos reappeared from behind a hill and launched a barrage of Stinger missiles at the hapless helicopter. (Apparently the programmers had forgotten to remove that part of the infantry coding.)

The lesson? Objects are defined with certain attributes, and any new object defined in terms of an old one inherits all the attributes. The embarrassed programmers had learned to be careful when reusing object-oriented code, and the Americans left with a newfound respect for Australian wildlife. Simulator supervisors report that pilots from that point onward have strictly avoided kangaroos, just as they were meant to.

*Thanks to Craig Murphy for this one in response to Jon's article on page 41.*

## World-class training for Delphi professionals now in Ireland



**Brooks Associates Ireland**

Tel 01-832 0373

[brooksassociates@eircom.net](mailto:brooksassociates@eircom.net)

[www.brooksassociates.com](http://www.brooksassociates.com)