How to use the TListView OnCustomDrawXXX events

Contents

Why this article?

The *OnCustomDrawXXX* event handlers of Delphi's *TListView* can be useful to make minor changes to the appearance of a list view control. They let developers avoid having to owner draw the control if they only want to make a few tweaks to its appearance. Using the event handlers means that Delphi (or Windows) continue to take responsibility for drawing list items and dealing with the highlight etc.

However, a quick Google search shows that the *OnCustomDrawXXX* events are not well documented out on the net. So, having experimented with the events using the list view's report style I decided to document my findings – hence this article.

Some "ifs and buts"

Before launching into the article proper, there are a few observations that need to be made about the limitations of list view custom drawing and of Delphi's implementation of it via the *OnCustomDrawXXX* events.

Don't expect too much

The custom drawing support facility was added by Microsoft to enable developers to implement minor customisations of the list view control without having resort to owner drawing the whole control. The facility was not designed for major customisations.

So don't expect too much from the *OnCustomDrawXXX* events – don't push them too far! If you want perform a major customisation then you should owner-draw the control.

Delphi version differences

The code in this article has been tested with Delphi 4 and Delphi 7. There are subtle differences in behaviour between the two versions of the compiler that we need to take into account. These differences will be flagged up in the text. The code has not been tested with Delphi 5 or 6 so I am not clear when the behaviour changed.

Bugs in Delphi implementation

There can be some problems with font rendering when using some of solutions presented in this article. Investigations indicate that the problem appears to be with Delphi's implementation of the code that triggers *OnCustomDrawXXX* events. The problems seem to relate to the list view's *Canvas* property. You can get round some of the problems by using the Windows GDI API directly rather than depending on the *Canvas* property.

Overview

Before diving into some code we'll briefly discuss what the the different *OnCustomDrawXXX* events do.

OnCustomDraw

OnCustomDraw allows drawing on the background of the list view control. We should handle this event if we want to draw or paint anything on the control's background.

In this article we'll look at using this event handler to:

- Draw a background bitmap.
- Shade columns in the background of the control.

The event could also be used to paint the background colour of the control, but if we're using just a plain colour it's easier just to set the *Color* property of the list view and let Delphi handle the painting.

OnCustomDrawItem

Handling *OnCustomDrawItem* allows you to influence how a list item is drawn without having to perform the actual drawing yourself. Changes are made to the list view's *Canvas* property before Delphi (or Windows) draws the list item. For example, the font or font attributes can be changed, as can the brush colour used to paint the background of the list item.

Whatever changes are made are applied to the whole list item including the *Caption* and any *SubItems*. To demonstrate this event handler an example is provided where alternate list items are displayed in different colours. It is not advisable to directly paint any part of the list item. You should configure the control's *Canvas* and leave the painting to Delphi / Windows. Use owner-drawing if you want to paint all or part of the list item yourself.

OnCustomDrawSubItem

OnCustomDrawSubItem is triggered for each subitem of every list item. It allows sub items to be customised individually.

The event provides a *SubItem* parameter that identifies the column to be painted. Column zero is the list view's *Caption* column while columns >= 1 represent any subitems. To access the string associated with the *SubItems* property use *SubItems[SubItem-1]*, providing *SubItem* >= 1.

Once again this event should be used to configure the *Canvas* rather than to draw on it.

OnCustomDrawSubItem will only be called by Delphi if *OnCustomDrawItem* is also handled. So if you need *OnCustomDrawSubItem* without *OnCustomDrawItem* you must also create a do nothing *OnCustomDrawItem* event handler.

Delphi Version Differences

In Delphi 4 *OnCustomDrawSubItem* is called for each of the columns starting at column 0, the column containing the list item's *Caption*. However, in Delphi 7 the event is only called for true sub items, i.e. from column 1.

Therefore in Delphi 4 *OnCustomDrawSubItem* can be used to customise all the columns while in Delphi 7 we must use *OnCustomDrawItem* to customise column 0 and *OnCustomDrawSubItem* to customise the other columns. Note that the Delphi 7 approach also works for Delphi 4.

Some rules

Putting the above together we get the following rules:

1. To paint the list view's background, handle the *OnCustomDraw* event.
2. To configure the painting of a whole list item (a row in a report style list view). Handle the *OnCustomDrawItem* event.
3. To configure the painting of all the "columns" in a list item separately (i.e. the caption and all the visible sub items), first handle the *OnCustomDrawItem* event to configure how column 0 (the *Caption*) is to be displayed and then handle *OnCustomDrawSubItem* to configure the other cells.

   Here is some boilerplate code to use when handling both *OnCustomDrawItem* and *OnCustomDrawSubItem*:

```
procedure TForm1.ListView1CustomDrawItem(Sender: TCustomListView;
  Item: TListItem; State: TCustomDrawState;
  var DefaultDraw: Boolean);
begin
  // Process column 0 (the Caption) here
end;

procedure TForm1.ListView1CustomDrawSubItem(Sender: TCustomListView;
  Item: TListItem; SubItem: Integer; State: TCustomDrawState;
  var DefaultDraw: Boolean);
begin
  // Ensure we ignore SubItem = 0 (Delphi 4)
  if SubItem = 0 then Exit;
  // Process column 1 and higher here
end;
```

Listing 1

Examples

Most of the rest of this article is spent in looking at some examples of what we can do by handling the *OnDrawItemXXX* events of list view controls that have the report style.

***ListView_XXX* Routines**

Several of these examples use *ListView_XXX* routines. These routines are Delphi implementations of the C "macros" defined in Microsofts's Windows header files. They are provided in Delphi's `CommCtrl` unit. So make sure you add `CommCtrl` to your **uses** clause otherwise some examples will not compile.

Displaying a background bitmap

To display the bitmap we need only handle the *OnCustomDraw* event. In this example we will tile a bitmap in the display. To tile the bitmap we have to calculate the list view's background area and offset the bitmap to allow for any scrolling and the size of any header. Much of the code of the event handler is devoted to calculating these offsets. The code is presented in *Listing 2* below.

```
procedure TForm1.ListView1CustomDraw(Sender: TCustomListView;
  const ARect: TRect; var DefaultDraw: Boolean);

  function GetHeaderHeight: Integer;
```

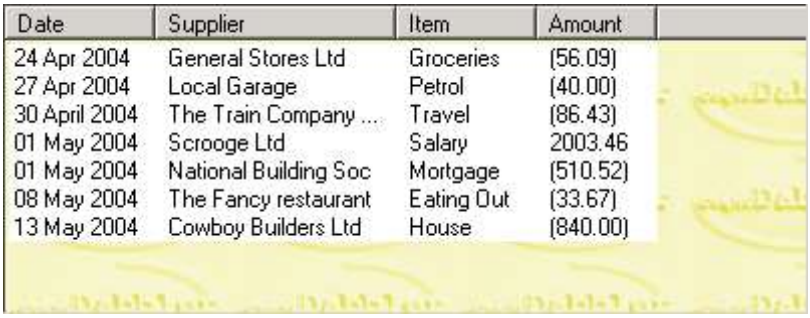
```
  var
    Header: HWND;           // header window handle
    Pl: TWindowPlacement;   // header window placement
  begin
    // Get header window
    Header := SendMessage(ListView1.Handle, LVM_GETHEADER, 0, 0);
    // Get header window placement
    FillChar(Pl, SizeOf(Pl), 0);
    Pl.length := SizeOf(Pl);
    GetWindowPlacement(Header, @Pl);
    // Calculate header window height
    Result  := Pl.rcNormalPosition.Bottom - Pl.rcNormalPosition.Top;
  end;

var
  BmpXPos, BmpYPos: Integer;  // X and Y position for bitmap
  Bmp: TBitmap;               // Reference to bitmap
  ItemRect: TRect;            // List item bounds rectangle
  TopOffset: Integer;         // Y pos where bmp drawing starts
begin
  // Get top offset where bitmap drawing starts
  if ListView1.Items.Count > 0 then
  begin
    ListView_GetItemRect(ListView1.Handle, 0, ItemRect, LVIR_BOUNDS);
    TopOffset := ListView_GetTopIndex(ListView1.Handle) *
      (ItemRect.Bottom - ItemRect.Top);
  end
  else
    TopOffset := 0;
  BmpYPos := ARect.Top - TopOffset + GetHeaderHeight;
  // Draw the bitmap
  // get reference to bitmap
  Bmp := Image1.Picture.Bitmap;
  // loop until bmp is past bottom of list view
  while BmpYPos < ARect.Bottom do
  begin
    // draw bitmaps across width of display
    BmpXPos := ARect.Left;
    while BmpXPos < ARect.Right do
    begin
      ListView1.Canvas.Draw(BmpXPos, BmpYPos, Bmp);
      Inc(BmpXPos, Bmp.Width);
    end;
    // move to next row
    Inc(BmpYPos, Bmp.Height);
  end;
end;
```

Listing 2

The event handler's *ARect* parameter provides the client area of the list view. Any header control is included in the client area, so the top of the display area for the list items begins at an offset equal to the height of the header control. We find the required height by calling the *GetHeaderHeight* subsidiary function. This routine gets the header window handle, retrieves its display rectangle and works out the height of the header from this. We also need to adjust the offset of the bitmap to allow for any scrolling in the list view. We get the index of the top item in the list and multiply this by the height of a single list item.

Having calculated the starting Y-offset for the image (and stored it in *BmpYPos*) we draw the bitmaps by using two nested loops. The outer loop displays the rows of bitamp, updating the Y-offset by the height of the bitmap. This loop terminates when the Y-offset is beyond the bottom of *ARect*. The inner loop handles tiling the bitmap across a row. We use *BmpXPos* to store the next X coordinate of the bitmap, beginning at *ARect.Left* and ending when *BmpXPos* goes beyond the right edge of *ARect*.

Delphi / Windows take care of displaying the list items after the background is drawn. List items are drawn with solid backgrounds, overwriting the newly drawn background, as can be seen in *Figure 1* below:



Date	Supplier	Item	Amount
24 Apr 2004	General Stores Ltd	Groceries	(56.09)
27 Apr 2004	Local Garage	Petrol	(40.00)
30 April 2004	The Train Company ...	Travel	(86.43)
01 May 2004	Scrooge Ltd	Salary	2003.46
01 May 2004	National Building Soc	Mortgage	(510.52)
08 May 2004	The Fancy restaurant	Eating Out	(33.67)
13 May 2004	Cowboy Builders Ltd	House	(840.00)

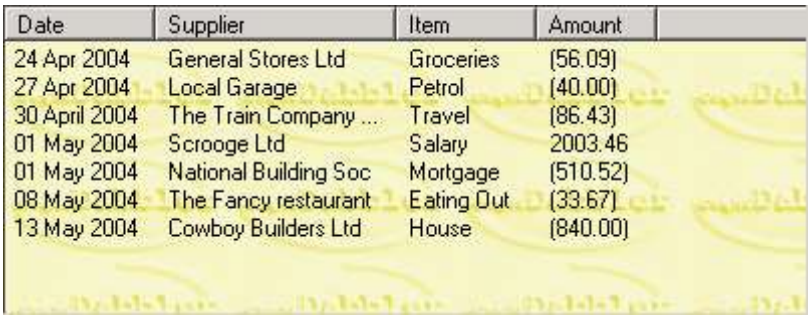
Figure 1

But what if we want the list items to appear transparently over the bitmap? We simply add the code shown in *Listing 3* to the end of the *OnCustomDraw* event handler from *Listing 2*:

```
  // Ensure that the items are drawn transparently
  SetBkMode(ListView1.Canvas.Handle, TRANSPARENT);
  ListView_SetTextBkColor(ListView1.Handle, CLR_NONE);
  ListView_SetBKColor(ListView1.Handle, CLR_NONE);
```

Listing 3

This new code ensures list items are drawn with tranpsarent backgrounds. The effect is shown *Figure 2*.



Date	Supplier	Item	Amount
24 Apr 2004	General Stores Ltd	Groceries	(56.09)
27 Apr 2004	Local Garage	Petrol	(40.00)
30 April 2004	The Train Company ...	Travel	(86.43)
01 May 2004	Scrooge Ltd	Salary	2003.46
01 May 2004	National Building Soc	Mortgage	(510.52)
08 May 2004	The Fancy restaurant	Eating Out	(33.67)
13 May 2004	Cowboy Builders Ltd	House	(840.00)

Figure 2

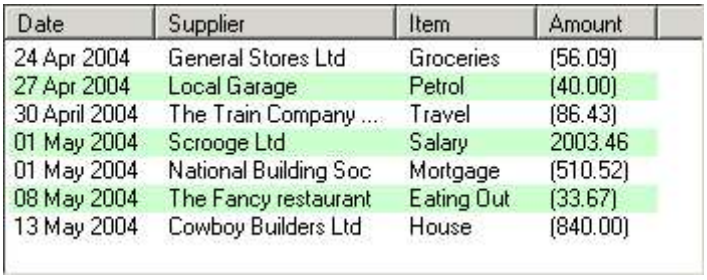
Drawing rows in alternating colors

This example is much simpler than the preceding one. We will draw alternating list items in different colours, emulating green and white line printer paper. This is achieved by handing just the *OnCustomDrawItem* event as *Listing 4* shows:

```
procedure TForm1.ListView1CustomDrawItem(Sender: TCustomListView;
  Item: TListItem; State: TCustomDrawState;
  var DefaultDraw: Boolean);
const
  cStripe = $CCFFCC;  // colour of alternate list items
begin
  if Odd(Item.Index) then
    // odd list items have green background
    ListView1.Canvas.Brush.Color := cStripe
  else
    // even list items have window colour background
    ListView1.Canvas.Brush.Color := clWindow;
end;
```

Listing 4

The code is quite simple. We simply check the list item's index. If it is odd we ensure that the list item's background is green, otherwise we use the system's window colour for the background. The required background colour is specified by setting the colour of the list view canvas' brush. *Figure 3* shows the resulting list view:



Date	Supplier	Item	Amount
24 Apr 2004	General Stores Ltd	Groceries	(56.09)
27 Apr 2004	Local Garage	Petrol	(40.00)
30 April 2004	The Train Company ...	Travel	(86.43)
01 May 2004	Scrooge Ltd	Salary	2003.46
01 May 2004	National Building Soc	Mortgage	(510.52)
08 May 2004	The Fancy restaurant	Eating Out	(33.67)
13 May 2004	Cowboy Builders Ltd	House	(840.00)

Figure 3

Drawing columns in different colours

Now we move on to demonstrate the *OnCustomDrawSubItem* event handler. Here we will display each list view column in a different colour. Again, we use a four column list view in report mode. *Listing 5* has the code:

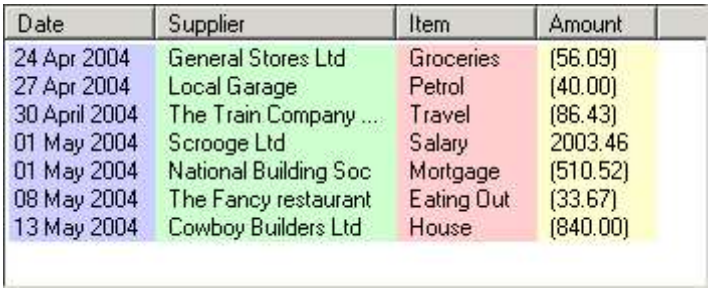
```
procedure TForm1.SetLVColumnColour(ColIdx: Integer);
  // Sets the list view brush colour for the column
const
  // The colours for each list view column
  cRainbow: array[0..3] of TColor = (
    $FFCCCC, $CCFFCC, $CCCCFF, $CCFFFF
  );
begin
  ListView1.Canvas.Brush.Color := cRainBow[ColIdx];
end;

procedure TForm1.ListView1CustomDrawItem(Sender: TCustomListView;
  Item: TListItem; State: TCustomDrawState;
  var DefaultDraw: Boolean);
  // Draw the "Caption" column
begin
  // Set the colour for column 0
  SetLVColumnColour(0);
end;

procedure TForm1.ListView1CustomDrawSubItem(Sender: TCustomListView;
  Item: TListItem; SubItem: Integer; State: TCustomDrawState;
  var DefaultDraw: Boolean);
  // Draw the sub item columns
begin
  // Check if SubItem is 0 and exit (Delphi 4 calls this event
  // with SubItem = 0, while Delphi 7 starts with SubItem = 1
  if SubItem = 0 then Exit;
  // We set the background colour to the colour required for
  // the column per the SubItem parameter
  SetLVColumnColour(SubItem);
end;
```

Listing 5

Firstly we declare a private method, *SetLVColumnColour*, to set the colour of a given column to some preset value.

We handle the *OnCustomDrawItem* event to set the colour for the *Caption* column, by calling *SetLVColumnColour* for column 0. This code sets the background for the whole list item, but we override this effect for each sub-item column in the *OnCustomDrawSubItem* event handler. Recall that this event handler is passed the index of the column to be painted in its *SubItem* parameter. We pass *SubItem* to *SetLVColumnColour* to set the background colour of the column. We have used the boilerplate code presented in *Listing 1* to ensure the code works with both Delphi 4 and Delphi 7: we ignore column 0 in *OnCustomDrawSubItem* because it has been dealt with in *OnCustomDrawItem*.

The resulting list view is shown in *Figure 4*:



Date	Supplier	Item	Amount
24 Apr 2004	General Stores Ltd	Groceries	(56.09)
27 Apr 2004	Local Garage	Petrol	(40.00)
30 April 2004	The Train Company ...	Travel	(86.43)
01 May 2004	Scrooge Ltd	Salary	2003.46
01 May 2004	National Building Soc	Mortgage	(510.52)
08 May 2004	The Fancy restaurant	Eating Out	(33.67)
13 May 2004	Cowboy Builders Ltd	House	(840.00)

Figure 4

Using different fonts in different columns

We can make each column use a different font or font style by handling the *OnCustomDrawItem* and the *OnCustomDrawSubItem* events. In this example we display the list item's *Caption* ("Date" column) in Comic Sans MS italic. We then present the remaining sub items in the standard list item's font, except that any negative values in the "Amount" column are to be displayed in red. *Listing 6* shows the code:

```
procedure TForm1.ListView1CustomDrawItem(Sender: TCustomListView;
  Item: TListItem; State: TCustomDrawState;
  var DefaultDraw: Boolean);
  // Set column 0 to use Comic Sans MS italic
begin
  ListView1.Canvas.Font.Name := 'Comic Sans MS';
  ListView1.Canvas.Font.Style := [fsItalic];
end;

procedure TForm1.ListView1CustomDrawSubItem(Sender: TCustomListView;
  Item: TListItem; SubItem: Integer; State: TCustomDrawState;
  var DefaultDraw: Boolean);
begin
  // Ensure SubItem 0 not updated here in Delphi 4
  if SubItem = 0 then Exit;
  if (SubItem = 3) and (Pos('(', Item.SubItems[2]) > 0) then
    // Display negative values in "Amount" column in red
    ListView1.Canvas.Font.Color := clRed
  else
    // Display all other sub item colums in black
    ListView1.Canvas.Font.Color := clBlack;
end;
```

Listing 6

Once again we modify the list view's *Canvas* property to get the desired result. We use the *OnCustomDrawItem* event handler to set the font required for column 0. The *OnCustomDrawSubItem* event handler then sets the font for sub items (columns 1 to 3). In column 3 we detect negative numbers – they begin with a '(' character – and set the font to red whenever we find one. The resulting list view displays as shown in *Figure 5*:

Date	Supplier	Item	Amount
24 Apr 2004	General Stores Ltd	Groceries	(56.09)
27 Apr 2004	Local Garage	Petrol	(40.00)
30 April 2004	The Train Company Ltd	Travel	(86.43)
01 May 2004	Scrooge Ltd	Salary	2003.46
01 May 2004	National Building Soc	Mortgage	(510.52)
08 May 2004	The Fancy restaurant	Eating Out	(33.67)
13 May 2004	Cowboy Builders Ltd	House	(840.00)

Figure 5

Display a shaded column

The Explorer in Windows XP displays the sorted column in pale grey. In this final example we mimic Explorer's behaviour by handling all three *OnCustomDrawXXX* events to display a list view with a specified column shaded in pale grey. For the purposes of this example we don't implement sorting, we just shade a column when its header is clicked.

In addition to the *OnCustomDrawXXX* event handlers we also need to handle clicks on the column headers in the list view's *OnColumnClick* event handler. *Listing 7* shows the *OnColumnClick* event handler:

```
procedure TForm1.ListView1ColumnClick(Sender: TObject;
  Column: TListColumn);
  // Handles column click: updates shaded column
begin
  // Updates the index of the shaded column
  fCurrentCol := Column.Index;
  // Redisplays list view with new column highlighted
  ListView1.Invalidate;
end;
```

Listing 7

This code simply records the index of the clicked column in a private form field and then invalidates the list view to redisplay it with the newly selected column highlighted. The private field is also used by the *OnCustomDrawXXX* event handlers and is defined in the form class definition as show in *Listing 8*:

```
private
  fCurrentCol: Integer; // currently selected column
```

Listing 8

At first sight in would appear that we merely need to handle the *OnCustomDrawItem* and *OnCustomDrawSubItem* events to set the column shading, in a similar manner to *Listing 5*. However a close examination of *Figure 4* shows that the column shading does not extend down to the bottom of the list view – it ends at bottom of the last list item. There is also a margin between the header control and the first list item. We want the shading to occupy the whole column from the header control to the bottom of the list view control.

To overcome this problem we handle the *OnCustomDraw* event to draw the column shading from the top to bottom of list view's client area. *Listing 9* defines the required *OnCustomDraw* event handler:

```
procedure TForm1.ListView1CustomDraw(Sender: TCustomListView;
  const ARect: TRect; var DefaultDraw: Boolean);
  // Displays shading in any area not occupied by list items
var
  ColLeft: Integer; // left edge of selected column
  ColBounds: TRect; // bounds of the selected column
  I: Integer;       // loops thru columns
begin
  // Calculate left side of selected column
  ColLeft := ARect.Left;
  for I := 0 to Pred(fCurrentCol) do
    ColLeft := ColLeft + ListView_GetColumnWidth(ListView1.Handle, I);
  // Calculate bounding rectangle of selected column
  ColBounds := Rect(
    ColLeft,
    ARect.Top,
    ColLeft + ListView_GetColumnWidth(ListView1.Handle, fCurrentCol),
    ARect.Bottom
  );
  // Shade the column
  // Delphi / Windows will overwrite this where there are list items but
  // this code ensures shading extends from top to bottom of the client
  // rectangle
  ListView1.Canvas.Brush.Color := cShade;
  ListView1.Canvas.FillRect(ColBounds);
end;
```

Listing 9

First we locate the position of the left hand edge of the selected column and store it in *ColLeft*. This is done by beginning at the left hand edge of the list view's display and adding the widths of all columns that precede the selected column, if any.

Next we calculate the bounds of the selected column. The bounding rectangle has top and bottom the same as those of the list view's display rectangle (per the *ARect* parameter). The left of the column is given by *ColLeft* and its width can be found by using the *ListView_GetColumnWidth* API call. Once we have the bounding rectangle of the selected column we fill it with the shading colour.

If we left it at that we would find that when Delphi / Windows draws the list items the shading under them would be overwritten. We overcome this by handling *OnCustomDrawItem* and *OnCustomDrawSubItem* to ensure that each column has the correct background colour. *Listing 10* shows the code for the event handler, which by now should be very familiar with *OnCustomDrawItem* handling column 0 and *OnCustomDrawSubItem* handling columns >=1.

```
procedure TForm1.SetLVColumnShading(ColIdx: Integer);
begin
  if fCurrentCol = ColIdx then
    // given column is selected: shade it
    ListView1.Canvas.Brush.Color := cShade
  else
    // given column not shaded: ensure correct background used
    ListView1.Canvas.Brush.Color := ColorToRGB(ListView1.Color);
end;

procedure TForm1.ListView1CustomDrawItem(Sender: TCustomListView;
  Item: TListItem; State: TCustomDrawState;
  var DefaultDraw: Boolean);
  // Shade the first column (index 0) if this is selected
begin
  SetLVColumnShading(0);
end;

procedure TForm1.ListView1CustomDrawSubItem(Sender: TCustomListView;
  Item: TListItem; SubItem: Integer; State: TCustomDrawState;
  var DefaultDraw: Boolean);
  // Shade a sub item column if selected
begin
  if SubItem > 0 then // ensure not column 0 (Delphi 4)
    SetLVColumnShading(SubItem);
end;
```

Listing 10

Similarly to *Listing 5* we use a helper method – this time *SetLVColumnShading* – to set the required colours. This method is passed a column index and checks to see if the column is the selected one. If so it sets the background colour to the shading colour, otherwise it sets the background colour to the list view's *Color* property. The event handlers should be self-explanatory by now and won't be discussed further.

Figure 6 has a picture of the resulting display after clicking the "Item" column header:

Date	Supplier	Item	Amount
24 Apr 2004	General Stores Ltd	Groceries	(56.09)
27 Apr 2004	Local Garage	Petrol	(40.00)
30 April 2004	The Train Company ...	Travel	(86.43)
01 May 2004	Scrooge Ltd	Salary	2003.46
01 May 2004	National Building Soc	Mortgage	(510.52)
08 May 2004	The Fancy restaurant	Eating Out	(33.67)
13 May 2004	Cowboy Builders Ltd	House	(840.00)

Figure 6

Demo program

A demo program that accompanies this article is available for download. It implements all the examples presented here.

This source code is merely a proof of concept and is intended only to illustrate this article. It is not designed for use in its current form in finished applications. The code, which is placed in the public domain, is provided on an "AS IS" basis, WITHOUT WARRANTY OF ANY KIND, either express or implied. If you agree to all this then please download the code using the following link.

Download the demo code

Summary

This article has summarised the purpose of the three list view *OnCustomDrawXXX* event handlers and has gone on to present several examples of using the event handlers with a report style list view.

The flaws in the Delphi implementation of the *OnCustomDrawXXX* event handlers were pointed out. The article also noted that the events are only intended for lightweight customisations of the list view. Owner drawing should continue to be used for major customisations.

A demo program that demonstrates the examples has also been made available.

Feedback

I hope the article provided you with some ideas of how the event handlers can be used in your own code. If you have any comments or observations, please *get in touch*.
