Grid and histogram controls for Windows implemented in Forth

N.J. Nelson

Abstract

Grid controls are used extensively for presenting tabular data in Windows applications. A wide variety of third-party grid controls are readily available, but none were quite suitable for the application we required. Graphical controls are also available, but again, none quite fitted the bill. Finally we plucked up courage and wrote our own controls. This paper will outline some of the code design, and illustrate the use of the controls in an application. The code is in MPE ProForth for Windows.

N.J. Nelson B.Sc., C.Eng., M.I.E.E.
Micross Electronics Ltd.,
Units 4-5, Great Western Court,
Ross-on-Wye, Herefordshire.
HR9 7XP U.K.
Tel. +44 1989 768080
Fax. +44 1989 768163
Email. njn@micross.co.uk

First, an apology

I intended to present a paper very similar to this one at an earlier conference. Pressure of work prevented me from doing so.

Introduction

All of the four applications that we produce for Microsoft Windows (“Tracknet”, “Rabit”, “Frames” and “Bender”) require the entry and presentation of data in tabular form. In most cases, the format of presentation is fixed and can be specified at design time. The entry and display methods are also straightforward, and can use standard Windows controls (editboxes, checkboxes etc.) for editing. Until recently, we have relied upon the custom controls provided by Simple Software Inc., under the name “WinWidgets”, to present the tabular controls. (This product is now obsolete.)

Recently, the requirements of presentation have become more complex. The table formats are configurable in run time, not at design time. The data presentation methods are more varied. The marketing department demands a product that is more distinctive and eye-catching.

A new requirement in some of our applications is that historical data must be presented as a histogram, or some other graphical form. Again, this must be configurable in run time.

Possible alternative products

There is a wide selection of grid controls available for use with Microsoft Windows. There is a smaller selection of histogram and other graphical controls available.

The main difficulty that the suppliers of these controls have is that they must be all things to all men. This makes the customisation of the controls a very complex business.

A further difficulty is that the interface with the majority of the newer controls is by the Microsoft Component Object Model (COM). This interface is not easy to support in Forth, because the data structures have been designed specifically to match Microsoft Visual C++. Very few controls are still available with Dynamic Link Library (DLL) interfaces, which could easily be linked to Forth.

Finally, the commercial controls are not available in source-code versions (or are only so available at great expense). This is anathema to the Forth programmer. While we are happy to use “black boxes” we must always retain the ability to open the black box when we really need to. There are very good reasons for this – all previous bought-in code that we have used has contained bugs, often of the most subtle and complex kind. With WinWidgets, for example, it was only when we obtained the source code that we were able to understand completely the work-arounds required for some of these bugs.

Finally, we decided that the only complete solution lay in producing our own grid and histogram controls.

Specification for Micross custom controls

1. The controls should become part of an application simply by adding the source code files to the include list for compilation (no DLL files or COM interfaces).
2. A design time interface must be provided for the Microsoft dialog editor, but it need only specify size and position.
3. The appearance and function of each control should be completely specified in a single data structure.
4. There should be a default action for every feature, so that the control can be customised only as required and on a step-by-step basis, in accordance with the tradition of Forth incremental development.

The design time interface

There are now only two types of control that are supported by the Microsoft dialog editor. Firstly there are Microsoft’s own controls, both the traditional type such the buttons, and the later and more complex additions such as the calendar control. Secondly there are the custom controls by both Microsoft and third party suppliers, which must use the COM interface.

Since we had decided that we would not use COM, the only solution was to use a standard Microsoft control for design time editing. Accordingly we selected the very simplest control, which is called Static. This control can be made to assume any desired appearance by using the SS_OWNERDRAW style, and it can be given any desired functionality by means of subclassing.

The structural description of a custom control

The structure shown below completely describes both the appearance and function of a grid control.

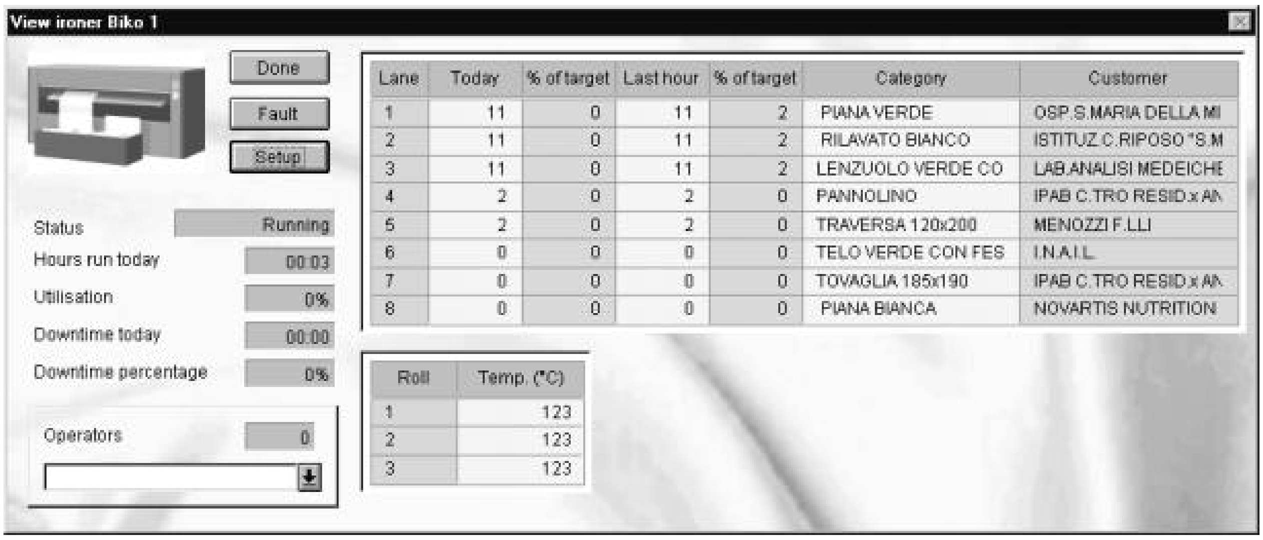
```
STRUCT MGRID
CELL    FIELD   MG.HWND               \ Handle of window
CELL    FIELD   MG.HDC                \ Device context
CELL    FIELD   MG.DCX                \ X position of grid within DC
CELL    FIELD   MG.DCY                \ Y position of grid within DC
CELL    FIELD   MG.CWIDTH             \ Client width
CELL    FIELD   MG.CHEIGHT            \ Client height
CELL    FIELD   MG.NCOLS              \ Number of columns
CELL    FIELD   MG.NROWS              \ Number of rows
CELL    FIELD   MG.BACKCOL            \ Background colour
CELL    FIELD   MG.HMARGIN%           \ Percentage of horizontal dimension which is
                                      \ margin
CELL    FIELD   MG.VMARGIN%           \ Percentage of vertical dimension which is
                                      \ margin
CELL    FIELD   MG.HMARGIN            \ Horizontal margin
CELL    FIELD   MG.VMARGIN            \ Vertical margin
CELL    FIELD   MG.GAP%               \ Percentage of vertical dimension which is gap
                                      \ between table and title
CELL    FIELD   MG.TIT%               \ Percentage of vertical dimension which is title
CELL    FIELD   MG.GAP                \ Gap between table and title
CELL    FIELD   MG.TITBACKCOL         \ Title background colour
CELL    FIELD   MG.TABBACKCOL         \ Table background colour
CELL    FIELD   MG.TITGRIDCOL         \ Title grid colour
CELL    FIELD   MG.TABGRIDCOL         \ Table grid colour
CELL    FIELD   MG.GRIDW              \ Width of grid
CELL    FIELD   MG.GRIDX              \ X coordinate of grid
CELL    FIELD   MG.TITH               \ Height of title block
CELL    FIELD   MG.TITY               \ Y coordinate of title block
CELL    FIELD   MG.TABH               \ Height of table block
CELL    FIELD   MG.TABY               \ Y coordinate of table block
CELL    FIELD   MG.FCOLX              \ Function for x coordinate of column n=0..
CELL    FIELD   MG.FROWY              \ Function for y coordinate of row n=0..
CELL    FIELD   MG.TITFONT            \ Handle of font for titles
CELL    FIELD   MG.TABFONT            \ Handle of font for table
CELL    FIELD   MG.FTITLE             \ Function for title of column n=0..
CELL    FIELD   MG.FTABLE             \ Function for text of column x=0.., row y=0..
CELL    FIELD   MG.TITTEXTCOL         \ Title text colour
CELL    FIELD   MG.TABTEXTCOL         \ Table text colour
CELL    FIELD   MG.TEXTTAB%           \ Horizontal dimension by which table text is
                                      \ tabbed (units of 0.1%)
CELL    FIELD   MG.TEXTTAB            \ Horizontal dimension by which table text is
                                      \ tabbed
CELL    FIELD   MG.FJUST              \ Function for justification of table data in
                                      \ column n
CELL    FIELD   MG.FSTYLE             \ Function for style of cell data in column n
CELL    FIELD   MG.FCOLBACKCOL        \ Function for column background colour
END-STRUCT
```

Initialisation of a custom control

When a dialog box is created, but before it appears on the screen, the Window Procedure (“WinProc”) of the dialog box is passed a WM_INITDIALOG message, in which the application must initialise all necessary features associated with the dialog.

In the case of a dialog box containing a Micross grid control, the necessary actions include allocating memory for grid control structure, setting the required parameters into that structure, then allowing to control to set all its other parameters to default values. Finally the simple and unsuspecting static control must be subclassed to give it the complex and sophisticated behaviour of a grid control.

The code examples come from an application which visualises a multilane commercial ironing machine.



Status	Running
Hours run today	00:03
Utilisation	0%
Downtime today	00:00
Downtime percentage	0%
Operators	0

Lane	Today	% of target	Last hour	% of target	Category	Customer
1	11	0	11	2	PIANA VERDE	OSP.S.MARIA DELLA MI
2	11	0	11	2	RILAVATO BIANCO	ISTITUZ.C.RIPOSO "S.M
3	11	0	11	2	LENZUOLO VERDE CO	LAB.ANALISI MEDEICHE
4	2	0	2	0	PANNOLINO	IPAB C.TRO RESID.x AN
5	2	0	2	0	TRAVERSA 120x200	MENOZZI F.LLI
6	0	0	0	0	TELO VERDE CON FES	I.N.A.I.L.
7	0	0	0	0	TOVAGLIA 185x190	IPAB C.TRO RESID.x AN
8	0	0	0	0	PIANA BIANCA	NOVARTIS NUTRITION

Roll	Temp. (°C)
1	123
2	123
3	123

(The client for this particular installation is in Italy, hence the obscure category and customer names.)

The dialog box is initialised as follows:

```
: IRONER-INITIALISE ( hwnd,mess,wparam,lparam---res )
  HDLG HIRONER !                    \ Save dialog handle
  MGRID ALLOCATE DROP IMTS !        \ Allocate main table structure
  MGRID ALLOCATE DROP IRTS !        \ Allocate roll table structure
  IRONER-SETTEXT                    \ Language specific text
  HDLG I2GCOMBO-ID HCTL HIGCOMBO !  \ Handle of grid combobox
  IRONER-HIDECOMBO                  \ Hide grid combobox
  IRONER-SAVESIZES                  \ Save original sizes and positions of controls
  IRONER-SETSIZES                   \ Set sizes and positions of controls to match
                                    \ configuration
  IRONER-SETDATA                    \ Set initial data
  -1 IROLDSTATUS ! IRONER-ANIMATE   \ Handle animation control
  4DROP TRUE
;
```

We must never forget to discard the allocated memory when the dialog box is closed, otherwise a memory leak will occur.

```
: IRONER-DESTROY ( hwnd,mess,wparam,lparam---res )
  IMTS @ FREE DROP            \ Free main table structure
  IRTS @ FREE DROP            \ Free roll table structure
  OK
;
```

Each static control which has to function as a grid control could be individually subclassed in the dialog initialise function above. However, in the example application, we have chosen to use “global subclassing”. This sets all instances a particular window class to have a modified behaviour. In this case, the only modification actually required is that the mouse must report its position within the control, so that the owner window can show, hide and move any edit control associated with the particular cell of the grid.

```
: (STAT-EX-WINPROC) { hwnd mess wparam lparam -- res } \ Extend to static winproc
  hwnd GWL_STYLE WINGETWINDOWLONG           \ Get window style
  SS_OWNERDRAW AND IF                       \ Check if owner drawn
    mess CASE
      WM_NCHITTEST OF                       \ Override the hit test message
        hwnd WINGETPARENT UM_CMLBD          \ Inform parent
        lparam hwnd GETDLGCTRLID WSMD       \ about coordinates & id
        HTTRANSPARENT
      ENDOF
      hwnd mess wparam lparam DEFSTATPROC   \ All other message use default handler
    ENDCASE
  ELSE                                      \ Any other style
    hwnd mess wparam lparam DEFSTATPROC     \ Use default handler
  THEN
; ASSIGN (STAT-EX-WINPROC) WINPROC STAT-EX-WINPROC
```

To accomplish the global subclassing, a window of the defined class must already exist. A dummy one pixel square window is therefore created upon which we can operate.

```
: SUPERSTAT ( --- ) \ Install the global subclassing - Call this from WM_CREATE
  NULL                                      \ exstyle
  "Static" ^NULL                            \ classname, winname
  SS_OWNERDRAW                              \ style
  0 0                                       \ x,y
  1 1                                       \ w,h
  CURR-HANDLE NULL WINAPPINST@ NULL         \ hparent, hmenu, instance, lparam
  WINCREATEWINDOWEX                         \ Create dummy window to get handle
  GCL_WNDPROC ['] STAT-EX-WINPROC >BODY     \ Set the new winproc
  REL>ABS WINSETCLASSLONG WPSTAT !          \ Save the old winproc
;
```

Painting the grid control

When the time comes to show the grid control on the screen, the owner dialog receives a WM_DRAWITEM message. Because there are two instances of the grid control in the example application, the dialog must first identify which grid control is required, before calling the correct paint function.

```
: IRONER-DRAWITEM ( hwnd,mess,wparam,lparam---res ) \ Owner draw call
  OVER CASE                                          \ According to ID of control
    I2TABLE-ID     OF IRONER-DRAWMAINTABLE   ENDOF   \ Main table static
  ( I2ROLLTEMP-ID )    IRONER-DRAWROLLTEMPS          \ Roll temperatures table
  ENDCASE
;
```

In the Windows Application Programming Interface (API), all drawing functions must specify the handle of the destination window or device context. Many years ago, we encapsulated all drawing functions, so that the applications programmer has no need to deal with handles, and can use the same high level functions on any type of print surface (e.g. the same function can be used for both a screen and a printer). This unfortunately results in a complication when responding to a WM_DRAWITEM message, which has different parameters from the normal WM_PAINT message.

```
: IRONER-DRAWMAINTABLE
{ hwnd mess wparam lparam | ops ocw pdis prs[ PRINTSURFACE ] -- res }
  lparam ABS>REL -> pdis                     \ Get pointer to draw item structure
  PRINTING @ -> ops 0 PRINTING !             \ Save print state & clear
  prs[ PRINTSURFACE ERASE                    \ Initialise window structure
  hwnd I2TABLE-ID HCTL prs[ !                \ Put handle into window structure
  pdis DIS.HDC @ prs[ WIN-HDC !              \ Put DC in window structure
  CURR-WINDOW -> ocw prs[ (CURR-WINDOW) !    \ Save current window & replace
  MAKEOBJECTS                                \ Create default objects
\ **************************
  pdis DIS.HDC @ pdis DIS.RCITEM             \ Draw sunken edge
  EDGE_SUNKEN BF_RECT WINDRAWEDGE DROP

  IRONER-INITMAINTABLE                       \ Initialise main table structure
  IMTS @ MGRID-DRAW                          \ Draw the grid
\ ***************************
  KILLOBJECTS                                \ Discard objects
  ocw (CURR-WINDOW) !                        \ Restore current window
  ops PRINTING !                             \ Restore printing state
  0                                          \ Result
;
```

We expect to hide all of this complexity (i.e. everything except the section between the stars) by encapsulation before we use the grid control again.

The word MGRID-DRAW is a highly complex function that uses all the parameters in the grid structure to produce the desired appearance. It is not appropriate to explain the exact details here, except to say that the function can be broken down into distinct sections.

```
: MGRID-DRAW { mg -- } \ Draw grid
  mg MGRID-DRAWBACK           \ Background
  mg MGRID-DRAWBLOCKS         \ Title and table blocks
  mg MGRID-DRAWGRIDS          \ Title and table grids
  mg MGRID-DRAWTITLES         \ Text for titles
  mg MGRID-DRAWCELLS          \ Table cell contents
;
```

Editing the grid control

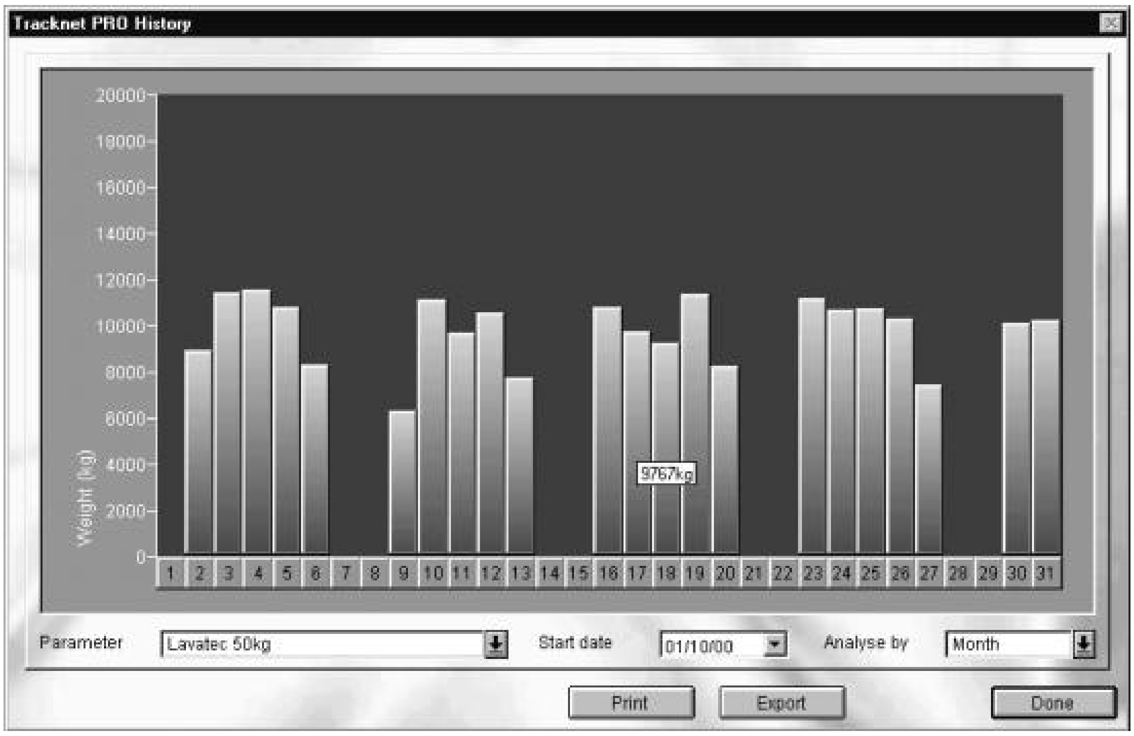
In most grid controls, a subsidiary editing window (editbox, checkbox, combobox etc.) is displayed whenever the user clicks the left mouse button within an editable cell of the grid. In our applications, however, most of the standard controls are already globally subclassed to operate on the left mouse button and (optionally) display an on-screen keyboard. These are used when a system has a touchscreen only (no keyboard) or when it is necessary to enter locale-dependent characters (e.g. French accented characters with only an English keyboard fitted). Therefore, in the grid control, the subsidiary editing window appears when the mouse hovers (without clicking) over the editable field.

The dialog box receives a user defined message from the grid control in response to the mouse hit test. After identifying the correct grid, the dialog box function adjusts and positions the subsidiary editing control, which in this case consists of a combobox filled with either category and customer names.

```
: IRONER-LBMTGRID { hwnd mess wparam lparam | col row -- res } \ Mouse move in table
  IMTS @ wparam LOWORD wparam HIWORD         \ Get coordinates
  IRONER-SCREENTOCLIENT                      \ Convert to client coordinates
  MGRID-XYTOCR -> row -> col                 \ Get column & row of grid
  col 5 6 WITHIN row -1 <> AND IF            \ Within the category & customer cells
    col row IRONER-FILLCOMBO                 \ Fill combo with required names
    row col IRONER-MOVECOMBO                 \ Position & size the grid combobox
    row col IRONER-SHOWSELECTION             \ Show the current item in the combobox
    row col IRONER-SHOWCOMBO                 \ Show grid combobox
    row HIGROW !
    col HIGCOL !
  ELSE                                       \ Not within the cat. or cus. cells
    IRONER-HIDECOMBO                         \ Hide the grid combobox
  THEN
  0                                          \ Result
;
```

The histogram control

The concept for the histogram control is very similar to that adopted for the grid control. A typical application of the control is shown below.



Parameter: Lavatec 50kg
Start date: 01/10/00
Analyse by: Month

The structure which completely describes the appearance and behaviour of a histogram is shown below:

```
STRUCT MGRAPH \ Generalised Micross Graph object
CELL    FIELD   G.HWND                \ Handle of window
CELL    FIELD   G.HDC                 \ Device context
CELL    FIELD   G.DCX                 \ X position of graph within DC
CELL    FIELD   G.DCY                 \ Y position of graph within DC
CELL    FIELD   G.CWIDTH              \ Client width
CELL    FIELD   G.CHEIGHT             \ Client height
CELL    FIELD   G.HFONT               \ Handle of horizontal annotation font
CELL    FIELD   G.HFONTV              \ Handle of vertical annotation font
CELL    FIELD   G.TEXTY               \ Address of text string for y axis
CELL    FIELD   G.MARGINX             \ Margin x axis
CELL    FIELD   G.MARGINY             \ Margin y axis
CELL    FIELD   G.NMARKSY             \ Number of marks on y axis
CELL    FIELD   G.NDIVSX              \ Number of divisions on x axis
CELL    FIELD   G.VALUE               \ Address of function ( n1=0..---n2 ) which
                                      \ returns value to plot for x axis item n1
CELL    FIELD   G.OFFSETY             \ Offset y axis
CELL    FIELD   G.SPANY               \ Span y axis
CELL    FIELD   G.ORIGX               \ Origin x coordinate
CELL    FIELD   G.ORIGY               \ Origin y coordinate
CELL    FIELD   G.FORMATY             \ Address of function which formats y axis
                                      \ annotation
CELL    FIELD   G.FORMATTIP           \ Address of function which formats tooltip
CELL    FIELD   G.FORMATPR            \ Address of function which formats value for
                                      \ printout
CELL    FIELD   G.MARKW               \ Width of marker (on x axis)
CELL    FIELD   G.TEXTEXT             \ Width of annotation string (on x axis)
CELL    FIELD   G.ANBLKH              \ Height of annotation block (on y axis)
CELL    FIELD   G.YAXISLEN            \ Length of y axis
CELL    FIELD   G.XAXISLEN            \ Length of x axis
CELL    FIELD   G.AXISCOL             \ Axis colour
CELL    FIELD   G.ABKGCOLX            \ Annotation background colour on x axis
CELL    FIELD   G.ANCOLY              \ Annotation colour (y axis)
CELL    FIELD   G.ANCOLX              \ Annotation colour (x axis)
CELL    FIELD   G.STRINGX             \ Address of function ( n=0..---z$ ) which
                                      \ returns annotation string on x axis
CELL    FIELD   G.VALCOL              \ Address of function ( n=0..---rgb ) which
                                      \ returns colour for value
CELL    FIELD   G.BACKCOL             \ Background colour
CELL    FIELD   G.GRCOL               \ Graph centre colour
END-STRUCT
```

Behaviour of a histogram control

This control clearly cannot be edited in the same way as a grid control, but it does use a mouse hover in a similar way, to generate a tooltip showing the value that each element of the histogram represents.

Licensing and copyright

In the "open code" spirit of Forth, I would be happy to provide source code for both the controls themselves and samples illustrating their use. Fellow Forth programmers are welcome to use the controls in their own applications, and in turn I would welcome suggestions for development and improvement. The actual applications in which I have used the controls must of course remain the copyright of Micross Electronics Ltd.

Conclusion

I hope that the above examples illustrate the power of Forth to harness some of the more intricate concepts of programming for Microsoft Windows.

References

1. The example programs are written in ProForth for Windows V2.1, by MicroProcessor Engineering Ltd., 133 Hill Lane, Southampton SO15 5AF U.K., Tel. 01703 631441

2. Readers wishing to thoroughly understand the inner workings of these controls, will need to subscribe to the Microsoft Developer Network (MSDN), full details of which will be found at http://msdn.microsoft.com/subscriptions/.