CenturaPro™

Visit us at www.ProPublishing.com! Hot Ideas for Centura® Developers

Killer Context Menus

R. J. David Burke

Learn how to make the most of Centura Team Developer's support for context menus and augment it with tabbed property dialogs.

Context menus have been around for some time in Windows, but it's in Windows 95 where Microsoft started providing first-class support for context menus. These are the floating, pop-up menus that present menu items specific to the selected object. The "programmer" documentation is fairly consistent in using the term "context menu," but you'll also find the terms "pop-up menu" and "shortcut menu" used interchangeably. The latter terms though are typically used in describing the Windows 95 user interface, as opposed to the Win32 API. I'll use the term "context menu" since that's the term used in Centura documentation. I recommend chapter 7 of the *Windows Interface Guidelines for Software Design* from Microsoft Press for more information.

Centura Team Developer (CTD) 1.0 includes support for context menus as part of its feature set. This article goes beyond the documentation, providing additional information on how to use context menus in your applications.

The context menu user interface

Typically, context menus are activated with a right click or, more correctly, by clicking an object with mouse button 2. On the down transition of the button, the object underneath the mouse pointer is selected. Following the up transition, the context menu displays.

The keyboard can also be used to display a context menu. With the appropriate keystroke, the object that has the focus is implicitly selected and its context menu displayed. Shift-F10 is the keystroke operation to launch the context menu. For the Microsoft keyboard and other compatible keyboards the Application key can also be used to launch a context menu. (The Application key is typically between the right Windows key and the right Shift key on the bottom row of the keyboard. It's labeled with a menu icon.)

Once a context menu is displayed, either the mouse or the keyboard (using the arrow keys) can be used to navigate the menu items and make a selection. Clicking the mouse or pressing the Enter key chooses a menu item, and its corresponding action is carried out. Clicking outside the menu or pressing the Esc key cancels the context menu operation.

Context menus for free

Adding basic context menus to an application requires no effort on the part of the developer. They're already integrated into all the "field" controls. By default, a system-defined context menu is displayed when you right click on a data field, multiline field, combo box, or table

Continues on page 3

July 1997

Volume 2, Number 7

PRO Publishing

What Do *You* Think?

Mark Hunter

Recently we asked *Centura Pro* readers to complete a survey to help us make the newsletter even more valuable to them. We were pleasantly surprised at the high response rate. Almost a fifth of the readership sent back the survey, about half by mail or fax, the rest via our Web site.

Three respondents will receive their own Voodoo PC! Carmine DiCostanzo at the Fish & Game Department in Alaska wins for submitting the very first response. A.J. Wood in the United Kingdom was the winning entry drawn at random from the hard-copy surveys we received. And Jos Meeuwsen in the Netherlands was chosen at random from electronic surveys filled out on the Pro Publishing Web site.

We were also surprised by some of the answers. Each issue of *Centura Pro* is read by three to four developers. Although we would be delighted to sell each of those individuals their own subscription (!), we are also pleased that issues have such a high readership—significantly higher than other similar publications.

Only five percent of respondents have "never" used the source code from the articles, but 60 percent have used it only "sometimes." We take that as a challenge! We'll keep trying to make the source code irresistible.

Readers would greatly prefer articles on Centura to those on SQLWindows in the coming year. There are a lot of SQLWindows applications running out there, but information on CTD and related products is strongly desired. Forty percent of survey respondents have already moved to CTD, and another 37 percent have firm plans to do so.

Also surprising was the huge number of respondents, 89 percent, who are already involved with Internet/intranet development or will be in the future. But only about half of those will be using CTD in that development. Close behind CTD is Java, with many other tools lagging behind those leaders.

What about SQLBase? Readers interested in SQLBase are definitely in the minority, but they seem to be a passionate minority. On the survey, few respondents had a "mid" level interest in SQLBase: We saw a lot of "high" answers and a lot of "low" ones. *Centura Pro* will continue to run occasional articles on SQLBase advanced topics. In the section where readers were encouraged to suggest their own topics, we also received requests for articles

Continues on page 16

Editor Mark Hunter, **Publisher** Dian Schaffhauser, **Business Manager** Shelley Doyle, **Production Editor** Paul Gould, **New Product** Addison Honor Gould (7 lbs., 4 oz.)

Centura Pro (ISSN: 1093-2100) is published monthly (12 times per year) by Pro Publishing, PO Box 18288, Seattle, WA 98118–0288.

POSTMASTER: Send address changes to *Centura Pro*, PO Box 18288, Seattle, WA 98118–0288.

Subscription information: To order, call Pro Publishing at 206-722-0406. Cost of domestic subscriptions: 12 issues, $119; Canada: 12 issues, $129. Other countries: 12 issues, $139. Ask about source code disk pricing. Individual issues cost $15. All funds must be in U.S. currency.

Centura technical support: Call Centura Software Corp. at 415-321-4484.

If you have questions, ideas for bribing authors, or would just love to chat about what you're doing with Centura products, contact us via one of the means at right.

Contact Us Here . . .

Centura Pro on the Web
http://www.ProPublishing.com

Editorial Command Post
Phone: 818-249-1364
Fax: 818-246-0487
E-mail: 71460.3142@compuserve.com

Subscription Headquarters
Phone: 206-722-0406
Fax: 206-760-9026
E-mail: Dschaffhauser@connect.com

Mail
Pro Publishing
PO Box 18288
Seattle, WA 98118-0288

Source Code on CompuServe
GO CENTURA, Library 10

Killer Context Menus . . .

Continued from page 1

window cell. The system-defined context menu provides working menu items for Undo, Cut, Copy, Paste, Delete, and Select All, as shown in **Figure 1**.

In the sample application that accompanies this article, the form window, frmDefault, lets you explore the built-in support for context menus. It also has one example of "preventing" context menus by returning FALSE to a SAM_ContextMenu message (in field df4). I've got more on SAM_ContextMenu in later sections of this article. An interesting anomaly to notice is that Paste is allowed on a non-editable combo box that always has its list dropped down. However, this appears to be a problem with Windows itself and not CTD.

Adding context menus to other controls

As part of the context menu support, the function SalContextMenuSetPopup was introduced in Centura Team Developer. This function is used to assign a Named Menu (in the outline) as the context menu for a specified window. It can be used to add a context menu to a control that doesn't have a system-defined context menu, such as a picture control, or it can be used to override the system-defined context menu for a "field" control. A typical use of this function is to call it during SAM_Create processing for a control. **Listing 1** provides an example, taken from the accompanying application.

Listing 1. Setting a context menu when a control is created.

```
♦ Picture: pic1
  ♦ Message Actions
    ♦ On SAM_Create
      ◊ Call SalContextMenuSetPopup( hWndItem,
          'menuPicture', 0 )
```

When calling SalContextMenuSetPopup, you'll always want to pass 0 for the third parameter unless you're using a table window, which I'll cover in the next section.

Another technique is to pop up the context menu yourself, calling SalTrackPopupMenu when a SAM_ContextMenu message is received, as shown in **Listing 2**.

Listing 2. Displaying a context menu programmatically.

```
♦ List Box: lb4
  ♦ List Initialization
  ♦ Message Actions
    ♦ On SAM_ContextMenu
      ◊ Call SalTrackPopupMenu( hWndItem,
        'menuList', TPM_ContextMenu, wParam, lParam )
```

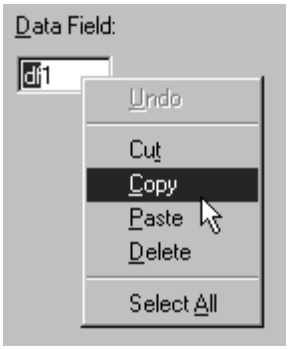

Figure 1. The standard context menu for character fields.

Notice that when a SAM_ContextMenu message is received, wParam and lParam have the mouse position (in screen coordinates). They can be used as is for a call to SalTrackPopupMenu to specify the origin coordinate of the pop-up menu. Also notice that the TPM_ContextMenu flag is used with SalTrackPopupMenu. Without this flag, you'd still have a floating pop-up menu, but the focus wouldn't be transferred to the right-clicked object.

SAM_ContextMenu is received by a control after it has been right clicked on (WM_RBUTTONDOWN and WM_RBUTTONUP), and before the context menu is about to be displayed. This message gives you the opportunity to:

- prevent a context menu from being displayed (by returning FALSE),
- specify a Named Menu within the application as the context menu (using SalContextMenuSetPopup),
- display a specified menu (using SalTrackPopupMenu), or
- perform other application specific tasks.

My preference is to set a default context menu on SAM_Create by calling SalContextMenuSetPopup. This has some advantages over directly calling SalTrackPopupMenu. When SalContextMenuSetPopup is used, it affects how WM_RBUTTONDOWN messages are handled. In a list box for example, the item under the mouse cursor is selected when you right click on it and SalContextMenuSetPopup has previously been called to define the floating pop-up menu to display.

If the default context menu (either provided by the system or assigned using SalContextMenuSetPopup) handles all cases, then there's usually no need to handle the SAM_ContextMenu message. However, you may want to handle the SAM_ContextMenu message to dynamically select a context menu based on the state of the selected object.

You can change the default context menu for a control with another call to SalContextMenuSetPopup, but more typically you call SalTrackPopupMenu to override the default menu for this message processing only. As you can see from the list boxes in frmOtherControls from the accompanying application, you have a lot of flexibility in how you process context menu requests. **Figure 2** shows some examples of context menus for non-field controls.

Finally, you can de-assign context menus by passing an empty string to SalContextMenuSetPopup. Notice that in frmOtherControls this has no effect on lb3 and lb4 because they're calling SalTrackPopupMenu on SAM_ContextMenu, regardless of whether a context menu is assigned to the list box.

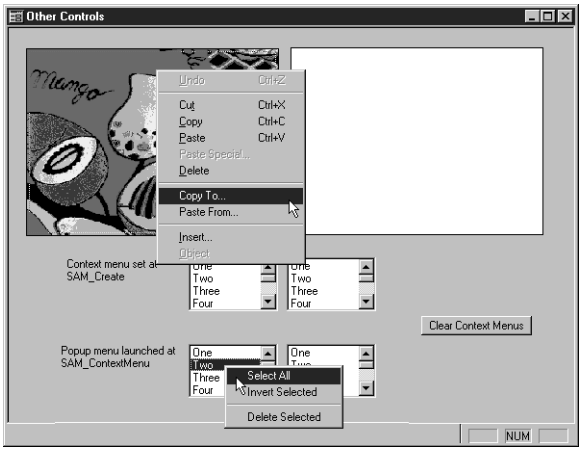

Figure 2. Some examples of context menus.

Context menus in table windows

Top-level and child table windows offer extended support for context menus. As I already mentioned, system-defined, default context menus are provided for cells. Not only can you replace the cell context menu, you can also define context menus for non-cell areas such as the row and column headers. This is where the third parameter of SalContextMenuSetPopup is relevant. Leave this parameter as 0 if you're defining a context menu for non-cell areas. If you're replacing the default context menu for cells, use the system defined constant CM_TableCell to indicate so.

The upper child table window in frmChildTables provides a simple example of context menus with table windows. Context menus for cell and non-cell areas are assigned during the window creation, as shown in **Listing 3**. They can be cleared by pressing the button at the top of the form.

Listing 3. Assigning simple context menus to a table window during its creation.

```
♦ Child Table: tbl1
      ♦ Contents
      ◊ Functions
      ◊ Window Variables
      ♦ Message Actions
          ♦ On SAM_Create
              ◊ Call SalContextMenuSetPopup( hWndItem,
                  'menuSimpleTable', 0 )
              ◊ Call SalContextMenuSetPopup( hWndItem,
                  'menuSimpleCell', CM_TableCell )
```

However, even more sophisticated context menu arrangements are possible. Suppose, for example, you wanted to have different context menus displayed when the row header or column header is displayed. Or you might want to display different context menus depending on whether the right click is in the upper or lower pane of a split table window. To support such arrangements, the new function SalTblObjectsFromPoint is used.

With SalTblObjectsFromPoint, you provide the coordinates of a point relative to the origin of the table window. The function returns the row and column that intersect at that point. Furthermore the function provides geographic information about the specified point, like whether it is in a row header or column header area, lower pane, split bar, or the like.

The idea, then, is to call SalTblObjectsFromPoint upon receipt of a SAM_ContextMenu message and determine the mouse pointer location within the table window to see what was right-clicked upon. Then the appropriate context menu can be displayed. The impedance mismatch is that SAM_ContextMenu provides the mouse location (via wParam and lParam) in screen coordinates, while SalTblObjectsFromPoint requires table window relative coordinates. The quick and easy solution is to call the Windows API function ScreenToClient. ScreenToClient converts screen coordinates to coordinates relative to a specified window.

After calling SalTblObjectsFromPoint, a Select Case or other conditional statement can be used to specify which named menu to use as the context menu for that location. The sample application includes table window classes that show how to use this function. **Listing 4** shows the specific implementation for the TableCM class.

I use a general window class, TableCM, to define the extended support for context menus with table windows. A top-level and child table window are both derived from TableCM for instantiating table windows with extended context menu support. In the SAM_Create message handler, I assign default context menus for both cell and non-cell contexts. Since this code is in a class, it doesn't make sense to refer to a named menu defined by a top-level window. Instead, use a named menu defined in global declarations. This provides the flexibility to instantiate the table window classes anywhere within

any appropriate scope of the application.

The SAM_ContextMenu handler starts by determining whether the message is the result of a mouse click or keyboard operation. If the keyboard interface was used, then some variables are fudged for the subsequent Select Case construct. If the mouse interface was used, then SalTblObjectsFromPoint is called to get the table component that was clicked on. (Other functions that may be useful for context menus include VisListGetIndexFromPoint, for list boxes, and cListView.HitTest, for list view controls.)

Next, a Select Case construct is used to launch an appropriate context menu. If the table is in cell mode, execution passes through to launch the assigned context menu. Otherwise, a specific context menu is launched via SalTrackPopupMenu for the clicked table component.

Now let me present one of the named menus I've defined as a context menu used with the TableCM class. **Listing 5** shows the context menu definition used when a column header is clicked on.

Note that the menu actions for all menu items issue a late-bound call to the appropriate function. This lets a derived class or an instance override the default behavior defined in TableCM. Where appropriate, a late-bound function call is also used to determine whether a menu item should be enabled. Using this approach tightly couples the menu definitions to the class definitions, which is desirable in this situation because the menus really are part of the class. The *Windows Interface Guidelines for Software Design* provides more detailed information on the content of context menus, such as how to pick appropriate and suitable menu items for the menu.

In designing the context menus for TableCM, I started with an

Listing 4. How to pick a context menu for a table window.

```
♦ General Window Class: TableCM
  ◊ Description:
  ◊ Derived From
  ◊ Class Variables
  ♦ Instance Variables
  ♦ Functions
  ♦ Message Actions
    ♦ On SAM_Create
      ◊ Call SalContextMenuSetPopup( hWndItem, 'menuTableEx', 0 )
      ◊ Call SalContextMenuSetPopup( hWndItem, 'menuCell', CM_TableCell )
    ♦ On SAM_ContextMenu
      ◊ Set i_bRowKeyboard = FALSE
      ♦ If wParam < 0 OR lParam < 0 ! context menu launched from keyboard
        ◊ Call SalTblQueryFocus( hWndItem, i_nFocusRow, i_hWndFocusColumn )
        ♦ If i_hWndFocusColumn = hWndNULL ! table is in row mode
          ♦ If SalTblQueryLockedColumns( hWndItem ) > 0
            ◊ Set i_nXFlag = TBL_XOverLockedColumns
          ♦ Else
            ◊ Set i_nXFlag = TBL_XOverUnlockedColumns
          ♦ If i_nFocusRow < 0
            ◊ Set i_nYFlag = TBL_YOverSplitRows
          ♦ Else
            ◊ Set i_nYFlag = TBL_YOverNormalRows
          ◊ Set i_nRow = i_nFocusRow
          ◊ Set i_bRowKeyboard = TRUE
      ♦ Else
        ◊ Set i_nX = wParam
        ◊ Set i_nY = lParam
        ◊ Call ScreenToClient( hWndItem, i_nX, i_nY )
        ◊ Call SalTblObjectsFromPoint( hWndItem, i_nX, i_nY,
                i_nRow, i_hWndCol, i_nFlags )
        ◊ Set i_nXFlag = i_nFlags & 0xFF00
        ◊ Set i_nYFlag = i_nFlags & 0x00FF
        ◊ Call SalTblQueryFocus( hWndItem, i_nFocusRow, i_hWndFocusColumn )
      ♦ Select Case i_nYFlag
        ♦ Case TBL_YOverNormalRows
          ♦ Select Case i_nXFlag
            ◊ Case TBL_XOverLockedColumns
            ♦ Case TBL_XOverUnlockedColumns
              ♦ If i_bRowKeyboard
                ◊ Call SalTrackPopupMenu( hWndItem, 'menuRow',
                                          TPM_ContextMenu, wParam, lParam )
                ◊ Return 0
              ♦ Else If i_hWndFocusColumn = hWndNULL
                ◊ Call SalTrackPopupMenu( hWndItem, 'menuTableEx',
                                          TPM_ContextMenu, wParam, lParam )
                ◊ Return 0
              ◊ Break
            ♦ Case TBL_XOverRowHeader
              ◊ Call SalTrackPopupMenu( hWndItem, 'menuRowHeader',
                                          TPM_ContextMenu, wParam, lParam )
              ◊ Return 0
              ◊ Break
            ♦ Default
              ◊ Return FALSE
          ◊ Break
        ♦ Case TBL_YOverSplitRows
          ♦ Select Case i_nXFlag
            ◊ Case TBL_XOverLockedColumns
            ♦ Case TBL_XOverUnlockedColumns
              ♦ If i_bRowKeyboard
                ◊ Call SalTrackPopupMenu( hWndItem, 'menuRow',
                                          TPM_ContextMenu, wParam, lParam )
                ◊ Return 0
              ♦ Else If i_hWndFocusColumn = hWndNULL
                ◊ Call SalTrackPopupMenu( hWndItem, 'menuTableEx',
                                          TPM_ContextMenu, wParam, lParam )
                ◊ Return 0
              ◊ Break
            ♦ Case TBL_XOverRowHeader
              ◊ Call SalTrackPopupMenu( hWndItem, 'menuRowHeader',
                                          TPM_ContextMenu, wParam, lParam )
              ◊ Return 0
              ◊ Break
            ♦ Default
              ◊ Return FALSE
          ◊ Break
        ♦ Case TBL_YOverColumnHeader
          ♦ Select Case i_nXFlag
            ◊ Case TBL_XOverLockedColumns
            ♦ Case TBL_XOverUnlockedColumns
              ◊ Call SalTrackPopupMenu( hWndItem, 'menuColumn',
                TPM_ContextMenu, wParam, lParam )
              ◊ Return 0
              ◊ Break
            ♦ Case TBL_XOverRowHeader
              ◊ Call SalTrackPopupMenu( hWndItem, 'menuTableAndHdr',
                TPM_ContextMenu, wParam, lParam )
              ◊ Return 0
              ◊ Break
            ♦ Default
              ◊ Return FALSE
          ◊ Break
```

examination of context menus in Excel and then considered the capabilities of the table window. For all TableCM context menus, I added Properties as the final menu item. (Before you flame me for not adding "…" after "Properties," here is my explanation. An ellipsis (…) is added after a menu item if the menu item is an imperative command and additional information is required to carry out the command. A typical example would be "Save As…". For a menu item that launches a dialog where the dialog is the essence of the command, no ellipsis is added. Such is the case with a Properties menu item.)

Visual Toolchest context menu anomalies

Generally, the Visual Toolchest classes and controls work with the context menu support provided in CTD. However, one thing that doesn't seem to be possible is setting a context menu for a drop-down control when it's in a drop-down state. This includes the calendar drop-down, the list view drop-down, and the color palette drop-down controls. From one point of view, this is consistent with Windows. For example, Windows itself doesn't support context menus for the list of a standard drop-down combo box. On the other hand, there may be a case for some deviation from Windows with the drop-down list view control.

Another Visual Toolchest anomaly is the Spin Field control class. While instances of this control do display the system defined context menu on a right click, they don't support the SalContextMenuSetPopup function (to replace the context menu) or the SAM_ContextMenu message.

Property dialogs

When the Properties item is selected from any TableCM context menu, it displays a tabbed properties dialog

Listing 5. The column context menu definition.

```
♦ Menu: menuColumn
  ◊ Title:
  ◊ Description:
  ◊ Enabled when:
  ◊ Status Text:
  ♦ Menu Item: Insert
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Insert column
    ♦ Menu Settings
      ◊ Enabled when: hWndItem.TableCM..CanInsertColumn(  )
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..InsertColumn(  )
  ♦ Menu Item: Delete All Columns
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Delete all columns
    ♦ Menu Settings
      ◊ Enabled when: hWndItem.TableCM..CanDeleteAllColumns(  )
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..DeleteAllColumns(  )
  ♦ Menu Item: Clear Contents
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Clear all cells in this column
    ♦ Menu Settings
      ◊ Enabled when: hWndItem.TableCM..CanClearColumnContents(  )
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..ClearColumnContents(  )
  ◊ Menu Separator
  ♦ Menu Item: Format...
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Explore format picture and input mask
    ♦ Menu Settings
      ◊ Enabled when:
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..ColumnFormat(  )
  ♦ Menu Item: Width...
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Explore width
    ♦ Menu Settings
      ◊ Enabled when:
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..ColumnWidth(  )
  ♦ Menu Item: Hide
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Hide this column
    ♦ Menu Settings
      ◊ Enabled when: hWndItem.TableCM..CanHideColumn(  )
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..HideColumn(  )
  ♦ Menu Item: Reveal
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Reveal all hidden columns
    ♦ Menu Settings
      ◊ Enabled when: hWndItem.TableCM..CanRevealColumns(  )
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..RevealColumns(  )
  ♦ Menu Item: Lock...
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Set number of locked columns
    ♦ Menu Settings
      ◊ Enabled when: hWndItem.TableCM..CanLockColumns(  )
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..LockColumns(  )
  ♦ Menu Item: Sort
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: Sort table on this column
    ♦ Menu Settings
      ◊ Enabled when: hWndItem.TableCM..CanSortByColumn(  )
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..SortByColumn(  )
  ◊ Menu Separator
  ♦ Menu Item: Properties
    ◊ Keyboard Accelerator: (none)
    ◊ Status Text: sal name, data type, cell type, list items, column title
    ♦ Menu Settings
      ◊ Enabled when:
      ◊ Checked when:
    ♦ Menu Actions
      ◊ Call hWndItem.TableCM..ColumnProperties(  )
```

(also known as a property sheet in the *Windows Interface Guidelines for Software Design*), where each tab is a logical grouping of related properties. There's some conscious design to the various property dialogs in TableCM. When the focus is in a cell and the Properties context menu item is selected, the properties dialog displays properties specific to the focus cell. When a column header is clicked and Properties is selected, the displayed property dialog presents four tabs of properties relevant to the column as a whole. If the row is clicked in the row header area, both row and row header properties are displayed on tabs in the properties dialog. When the click occurs in the upper left corner, various table and row header properties are displayed. And finally, when the context is a selected row, the table, column, row, and cell properties are accessible, as shown in **Figure 3**.

Since many of the same tabs appear on several different property dialogs, I was able to get some reusability by creating the tab contents as a form window and associating the form window with multiple tabs at design time. Another option would be to define one tabbed dialog that dynamically creates tabs and assigns windows to the tabs as required. Another possibility would be to follow the *Windows Interface Guidelines for Software Design* recommendations for putting a View drop-down list on the tabbed dialog to limit the tabs displayed, instead of the cluttered dialogs I've developed.

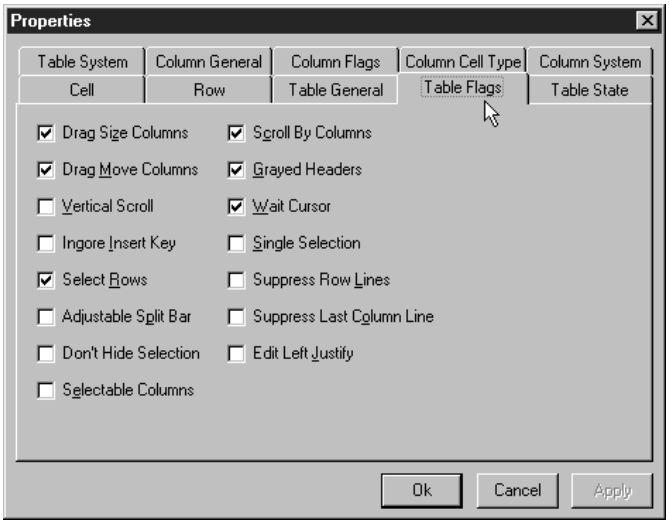

Figure 3. The many tabs available when choosing Properties from the row context menu.

Gems

Searching the source code of the TableCM class and the various property dialogs provides a variety of table window tips, tricks, and techniques, of which at least a few are undocumented. Grateful acknowledgment to SAL hacker extraodinaire, SAM User, who contributed extensively to the implementation of the context menu items and property dialogs. The remainder of this section discusses the gems you'll find in the demo application.

Listing 6 shows a fragment from the constants defined in the accompanying application. These constants are undocumented, but some have been revealed elsewhere (such as David Holmes-Kinsella's book, *Using SQLWindows 5* from Que and other *Centura Pro* articles) and some have been spread through word-of-mouth in the SAL community.

Listing 6. Undocumented table window flags.

```
◊ Number: COL_Visible = 0x00000001
◊ Number: COL_Inserted = 0x00000008
◊ Number: COL_FmtInvisible = 0x00000010
◊ Number: COL_GrayedTitle = 0x00000020
◊ Number: COL_NoChangeCaption = 0x00000040
◊ Number: COL_ReadOnly = 0x00000200
◊ Number: COL_LowerCase = 0x00000400
◊ Number: COL_UpperCase = 0x00000800
◊ Number: COL_FormView1Line = 0x00010000
◊ Number: COL_BiDi = 0x00400000
◊ Number: COL_Etched = 0x01000000
◊ Number: ROW_InMemory = 0x0001
◊ Number: ROW_Deleted = 0x0010
! ROW_UserFlags* defined by Visual Toolchest
◊ Number: ROW_CellAttribute = 0x1000
◊ Number: ROW_Inverted = 0x4000
◊ Number: ROW_DontIsolate = 0x8000
◊ Number: TBL_RowHdr_MapColumn = 0x0002
◊ Number: TBL_Flag_SelectRows = 0x0008
◊ Number: TBL_Flag_IgnoreInsert = 0x0010
◊ Number: TBL_Flag_AdjustSplitBar = 0x0020
◊ Number: TBL_Flag_DontHideSel = 0x0040
```

COL_FormView1Line affects how the column is displayed as a field when the table window is in form view. I believe that COL_NoChangeCaption and COL_Etched are used with columns when the table is displayed in form view. TBL_RowHdr_MapColumn doesn't seem to be implemented nor does TBL_Flag_SelectRows. SAL defines two user available row flags, and the Visual Toolchest adds three more.

In CTD 1.0 several new functions were added to SAL to define and query table column cell types. However, an apparent oversight is the lack of a function to define a column with a standard cell type. In the accompanying application, I define SalTblDefineStandardColumn to implement this by sending an undocumented message (WM_USER + 122) to the table. wParam contains the 0-based column identifier and lParam must contain 0. See **Listing 7** for the implementation.

Listing 7. Defining a column to have a standard cell type.

```
Function: SalTblDefineStandardColumn
  Description:
  Returns
    Boolean:
  Parameters
```

```
  Window Handle: p_hWndColumn
Static Variables
Local variables
  Boolean: bRetVal
  Window Handle: hWndParentTable
  Number: nColID
Actions
  Set bRetVal = FALSE
  If SalGetType( p_hWndColumn ) = TYPE_TableColumn
    Set hWndParentTable = SalParentWindow( p_hWndColumn )
    Set bRetVal = SalSendMsg( hWndParentTable, WM_USER + 122,
      SalTblQueryColumnID( p_hWndColumn ) - 1, 0 )
  Return bRetVal
```

In TableCM.DeleteAllColumns, SalTblDestroyColumns only works when all columns in a table window have been dynamically created. To determine whether to enable the corresponding menu items, I loop through all the columns and check to see that their SAL name is ''. Otherwise I know the column was defined at design time. Alternatively, I could have checked to see that all columns had the undocumented flag COL_Inserted set.

Adding a CopyTo function to copy selected rows to a CSV file is handy for exporting table window data. The original source for this idea was TeamAssist's Bud Ingraham.

One thing I couldn't determine was whether Select All should be enabled for a cell context menu. I couldn't find any technique, documented or undocumented, on how to get the character selection range within a cell.

Calling SalTblQueryFocus can be used to detect whether the table is in row mode or cell mode. If the hWndColumn parameter is hWndNULL, the table is in row mode.

Another undocumented table window message is WM_USER + 90. The return value is true or false, indicating whether a table window is a dynamic table window (SalTblSetRange with TBL_MaxRow for the range high value).

Finally, one way to detect whether a top-level window has accessories enabled is shown in **Listing 8**, taken from __frmTableProperties_System.OnInitialize.

Listing 8. Detecting whether accessories are enabled.

```
Set bAccEnabled = SalGetType( GetParent( hWndTable ) ) =
  TYPE_CustControl
```

Given the window handle of a top-level window, use the Windows API function GetParent to get the true parent window of the top-level window. If the return value from SalGetType on the parent window is equal to TYPE_CustControl, then the parent window is an accessory frame and accessories must have been enabled at design time for the top-level window.

The Windows Interface Guidelines for Software Design: An Application Design Guide

576 pages
$29.95
Microsoft Press
ISBN 1-55615-679-0

Building user-friendly applications

Context menus and property dialogs are important features in Windows applications. Users have come to expect and depend on these features in their day-to-day usage of 32-bit Windows applications. With the information provided in this article, you should be on your way to adding these features to your own applications. **CP**

Download CONTEXT_MENU.ZIP from www.ProPublishing.com or from library 10 in the Centura forum on CompuServe. You'll also find it on this month's Source Code Disk.

R.J. David Burke works in Centura Software's products group. He can be reached through the *Centura Pro* editorial staff.

In-place Editing of Child Window Titles

Centura Tip!

If you're designing the layout of your window and would like to edit background text, push/radio button titles and check box titles, just right-click your mouse while holding down the Shift key. You will then be able to edit the text directly in the design window. This works for both SQLWindows and CTD.—***Arne Bivrin***

Super-flexible Table Windows!

Gianluca Pivato

Create just about any type of table window you like. Use pictures, datafields, listboxes, radio buttons, and even child table windows to create your own super-flexible table window.

What I like most about Microsoft Access is the ability to use entire forms as rows in child windows. Access multiplies the child-form vertically, creating something like a table window. In this form you can use any child control that you would normally use on any other form. Well, you can do the same thing using SQLWindows—or Centura—and the result is amazing.

What is the Super-flexible Table Window?

The Table Window is a very useful and versatile control. You basically can't build a real application without using a table window of some sort. But sometimes the simple grid it exposes isn't adequate. The SFT (Super-flexible Table) allows you to design your own rows, header, and footer in the same way you would design any SQLWindows form. You can use any child window you like. (You could also put another table window inside a row; it wouldn't be practical but it *is* possible.)

The result is an Access-style table window. The SFT class creates the rows you designed, one below another, and simulates all the standard table window behavior. All the SalTbl* functions are available, and the content of the underlying table window's columns is transferred to and from the child windows in your rows. You can also switch from table-view mode to row-view mode and back.

How does it work?

There's a whole lot of CreateWindowEx( ) going on! The SFT is derived from a Child Table class, which makes it a table window. At creation time, the class creates a "static" window using the CreateWindowEx( ) function. This window has the same size and position as the table window. Then it creates the header, footer, and body forms as children of the "static" window. I don't show the "static" window in **Figure 1** because it would only make the figure more intricate; plus I use this window only as a container for scrolling and positioning purposes, and it doesn't have any code attached to it (it can't).

The header and footer (which are optional) are created from the dialogs you've designed from cSFTableHeader and cSFTableFooter and named after the SFT plus the "_Header" and "_Footer" suffixes. The body is the empty dialog created from cSFTableBody with the "_Body" suffix. The rows are created at different times, during the use of the control, as children of cSFTableBody. The rows are instances of the dialog that you have designed from cSFTableRow and (like all the other SFT components) named after the SFT plus the "_Row" suffix. Between the child table window and the SFT components there's a "Frame" window (not shown in the figure) used as the parent of all the SFT components and created using CreateWindowEx( ).

Class structure

The cSFTableWindow class is derived from the Table Window class and contains from zero to one (0,1) cSFTableHeader; from zero to 1 (0,1) cSFTableFooter and always one (1,1) cSFTableBody. The cSFTableBody class contains from zero to many (0,n) cSFTableRow. This means the header and footer are optional, and there can be zero or more rows contained in the table body. See

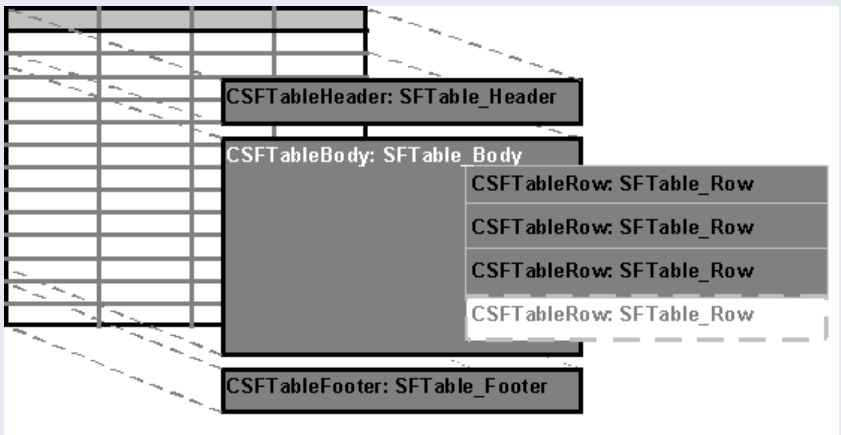

Figure 1. The SFT components relation.

Figure 2. All the SFT components (classes derived from cSFTableComponent) are Dialogs.

Inside the cSFTableWindow class

I create all the SFT components using the SalCreateWindowEx( ) function. The child table window (cSFTableWindow) you put in your form is the owner of the Header, Footer, and Body; the Body is the owner and the parent of all rows (each row is a dialog). This takes care of the creation, but there are some other problems to take care of as well: row scrolling, row buffering, binding a row's child windows to columns, and "dirty" detection.

Row scrolling

The SFT's rows are normal windows, and they're all children of cSFTableBody. If I didn't have to create and destroy the rows when they exceed the buffer limits and if Windows was able to scroll child windows for more than an integer value (32,767) the scrolling wouldn't be a problem at all. Unfortunately, that's not the case. There are two basic events to take care of: The scroll bar's thumb changes position, or the scroll bar changes its range after the SFT is resized.

When the thumb changes position, I find out how many rows have to scroll into view from the top or bottom. If that number is lower than the number of rows that can exist at the same time, I just scroll the cSFTableBody using ScrollWindow( ) and create all the missing rows, destroying all the scrolled-off rows. If the number of rows is greater than the row-buffer, then I destroy all the existing rows and recreate the new row view without using ScrollWindow( ). For example, if the buffer can hold 20 rows and the user scrolls 200 rows, then I destroy the 20 rows and recreate the new 20 rows. But if the user scrolls only two rows, I create the new rows (at the top or bottom according the scroll direction) and destroy the scrolled-off rows on the opposite end.

If I did this for a number of rows larger than what the buffer can hold, the result would be like a refresh of rows, where a large number of windows will be created and destroyed almost at the same time, at great expense in execution time.

When the SFT changes its size, the scroll bar range and thumb position have to be recalculated. Sometimes, usually when the SFT gets bigger, more rows have to come into view. I find out how many rows have to come into view by subtracting the position of the last row in the buffer from the total number of rows that can fit in cSFTableBody.

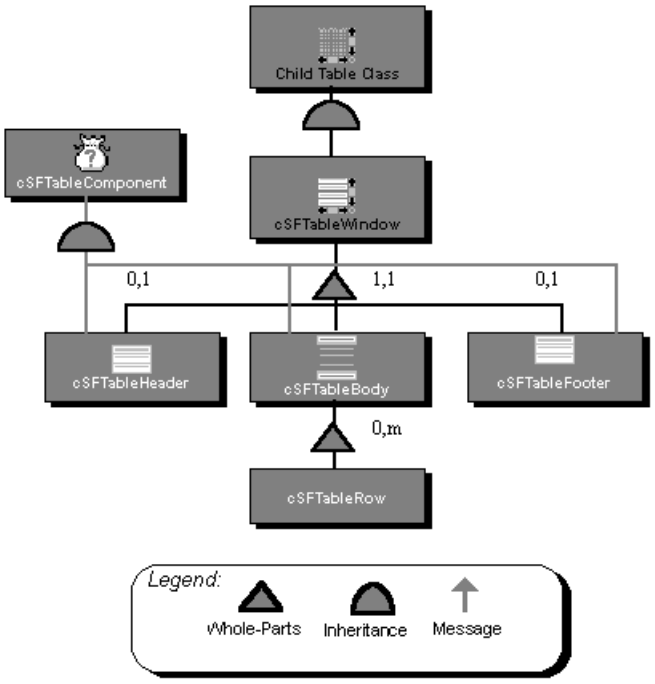

Figure 2. The SFT class structure.

Row buffering

A normal table window can have up to 32,767 rows. Obviously the SFT can't create 32,767 rows—which are entire dialogs with all their child windows—nor 10,000, nor even a few hundred. That would eat up all the PC's resources and make it almost impossible to browse the rows, since Windows would take forever to scroll all the dialog-rows inside the SFT Body.

The solution is to create as many rows as can fit on the screen and handle them in a "circular" list. The top item is deleted when a new one is added at the bottom and vice versa. The creation of a new row-window triggers the data transfer from the underlying columns to the row's child windows, just as the destruction of a row-window triggers the data transfer in the opposite direction. The buffer is only an array of window handles that can't have more than a given number of elements.

Table 1. The SFT equivalents for SalTbl* functions.

SAL Table Window Functions	SFT external equivalent	SFT internal equivalent
SalTblInsertRow	SFTInsertRow	InsertRow
SalTblDeleteRow	SFTDeleteRow	DeleteRow
SalTblDeleteSelected	SFTDeleteSelected	DeleteSelected
SalTblDoDeletes	SFTDoDeletes	DoDeletes
SalTblDoInserts	SFTDoInserts	DoInserts
SalTblDoUpdates	SFTDoUpdates	DoUpdates
SalTblPopulate	SFTPopulate	Populate
SalTblSetContext	SFTSetContext	SetContext
SalTblSetFocusCell	SFTSetFocusCell	SetFocusCell
SalTblPasteRows	SFTPasteRows	PasteRows
SalTblScroll	SFTScroll	Scroll
SalTblSetRange	SFTSetRange	SetRange

Binding a row's child windows to columns

Data has to go from the table window's columns to the child windows in your rows and back. The transfer has to take place when the view-mode changes, to keep the two

views synchronized and before calling any SalTbl* function that changes the content of the underlying table window. The detection of the view-mode change isn't a problem since it's done through an SFT service; but the only way to handle the use of SalTbl* is to write wrap-around functions in the SFT class.

The SFT moves data between the columns and the child windows using an internal "bind list" created at creation time. This list can be generated by you or by the SFT class. Prevent the SFT from creating the bind list by overwriting the SAM_Create message in your row dialog created from cSFTableRow, and adding bound controls using:

```
BoundToColumn( hWndChild, hWndColumn )
```

The SFT class generates the list using the child window's name. To bind a child window to a column, add the column name, prefixed by an underscore, at the end of the child window's name. Use the column ID to bind to dynamically-created columns:

```
Data Field: dfName_colName
DataField: dfName_#
```

The SFT class does its best to convert the data to match the child window's data type, but sometimes it's impossible to correctly detect the conversion to use—or you might want to use your own conversion. That's why, before changing a row's child window content, the child window receives the PAM_SFT_WriteData message and, before reading the row's child window content to write to the bound column, the child window receives the PAM_SFT_ReadData message. You can execute your conversion functions and return TRUE.

```
On PAM_SFT_WriteData
  Call <Your set data function>( GetChildData() )
  Return TRUE

On PAM_SFT_ReadData
  Call SetChildData( <Your get data function> )
  Return TRUE
```

The SFT assumes that the checked value of check boxes corresponds to "Y" and the unchecked to "N." It ignores radio buttons and pictures, so you have to take care of the data transfer for these two types of child windows using the technique described earlier.

Dirty detection

In a normal table window, whenever you edit a column's content, the ROW_Edited flag is automatically set. The SFT does it by intercepting the WM_COMMAND message sent from child windows to their parent. This technique allows you to detect when anything in your form changes state or value. For cSFTableRows I limited the detection to bound child windows. You can always set the dirty flag by calling:

```
cSFTableRow.SetDirty( bDirty )
```

A good moment to do this would be at the beginning or the end of the transfer process. You are notified of this event by PAM_SFT_Transfer:

```
PAM_SFT_Transfer
  wParam = {SFT_TransferBegin | SFT_TransferEnd } +
       {SFT_TransferToColumns |
       SFT_TransferFromColumns}
  lParam = Row number.
```

You can use this technique outside the SFT library to create a class that automatically detects its dirty state; that's much better than scanning all the child windows while testing the edit flag.

How to use the library

Put the SFT child in your form like any other control. Then create the SFT components from the four classes: cSFTableHeader; cSFTableFooter; cSFTableBody, and cSFTableRow. The header and footer are optional. The body and rows are required. The body doesn't need work; just create it. You could use it to show some background information as shown in the first sample application. The rows are completely open and available to your own needs and imagination. You can switch view using:

```
cSFTableWindow.SetView( {SFT_ShowTable |
  SFT_ShowRows} )
```

The SFT starts in row view, but you can change that by calling SetView( ) in SAM_Create. You can tell the SFT that you have header and/or footer components using:

```
cSFTableWindow.SetStyle( SFT_TableHasHeader |
  SFT_TableHasFooter )
```

. . . or that you want the SFT to create a border between the rows using:

```
cSFTableWindow.SetStyle( SFT_RowsHaveBorder )
```

If for any reason you want to force the data transfer to and/or from the columns, you can use:

```
cSFTableWindow.UpdateColumns()
cSFTableWindow.UpdateRows()
```

All the other SalTbl* functions are listed in Table 1. Just remember that you have to populate the SFT using Tbl_FillAll, in case you don't use cSFTableWindow.Populate( ).

Exploring the sample applications

I have provided two sample applications for this article. The first one is a little more useful than the second. The first example populates the SFT with all the picture files in a directory. You can select the directory in the cDirListBox child window. You can also switch from "Table view" to "Rows view" using the option buttons above the SFT. Zoom

the picture by double-clicking on it, as shown in **Figure 3**.

This sample application takes advantage of the PAM_SFT_WriteData message to set the picture with the relative file, and PAM_SFT_SetContext to change the background color of the row being selected. I intercept the message in the dfName_colName datafield to get the file name using GetChildData( ):

```
Data Field: dfName_colName
  On PAM_SFT_WriteData
    Call SalPicSetFile( picPicture,
      sftPic_Row.GetChildData() )

cSFTableRow: sftPic_Row
  On PAM_SFT_SetContext
    !Change the background color
    Call SetContext()
```

The second sample application is quite simple. I've added it to the article only to show how to use some basic SalTbl* functions and rather ugly color and font possibilities (which don't show up in all of their miserable glory on this two-color page), plus some other SFT features.

The application populates the SFT from table SYSCOLUMNS using:

```
Call sftTest.Populate(
      hSql,
      "select name, tblname, coltype, nulls
        from SYSCOLUMNS" )
```

The columns are created dynamically so I can also show how to bind a row's child windows to dynamic columns. The row template has a Pushbutton to the left that shows the row number. I set its value using:

```
cSFTableRow: sftTest_Row
  n PAM_SFT_Transfer
    if wParam & SFT_TransferBegin
      all SalSetWindowText( pbRowNumber,
          SalNumberToStrX( lParam, 0 ) )
```

The child windows are bound to their relative column using the column ID instead of the column name:

```
cSFTableRow: sftTest_Row
  Contents
    Data Field: df1_1
    Data Field: df2_2
    Data Field: df3_3
    Check Box: cb1_4
```

You can insert rows, delete rows, and scroll the table using the pushbuttons above the SFT. That's to show how the SFT updates the view when the rows are showing and you insert or delete a row. You can switch from "Table view" to "Row view" using the option buttons above the SFT.

Beyond the limits

Table windows are great tools. Talented developers have done much with them over the years. Super-flexible Table Windows will now let you work around the remaining user-interface limitations of regular table windows. Use this technique whenever you have a special need for a sophisticated interface. It's fast, easy, and impressive. **CP**

Download SFT.ZIP from www.ProPublishing.com or from library 10 in the Centura forum on CompuServe. You'll also find it on this month's Source Code Disk.

Gianluca Pivato, currently based in the U.S., is working on a project for second largest Italian Patents and Trademarks firm. The application is "an advanced data-warehousing/workflow-management/office-automation/professional-support system that's able to work in any country and to adapt its behavior to the intellectual property laws in any part of the world." In his spare time he studies "intelligent business applications" and writes drivers, word processors, games, language extensions, and scientific applications. Reach him in care of the *Centura Pro* editorial team.

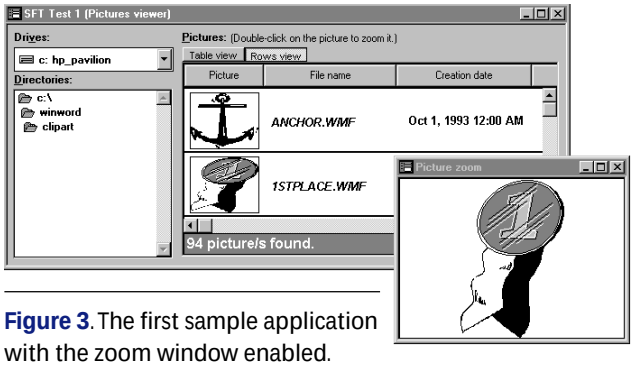

Figure 3. The first sample application with the zoom window enabled.

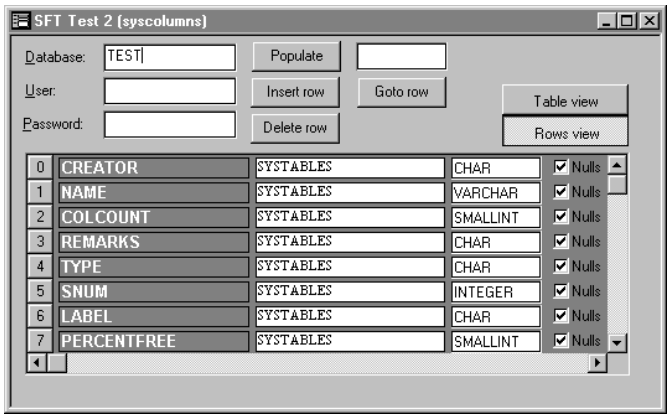

Figure 4. The second sample application showing SYSCOLUMNS in row-view.

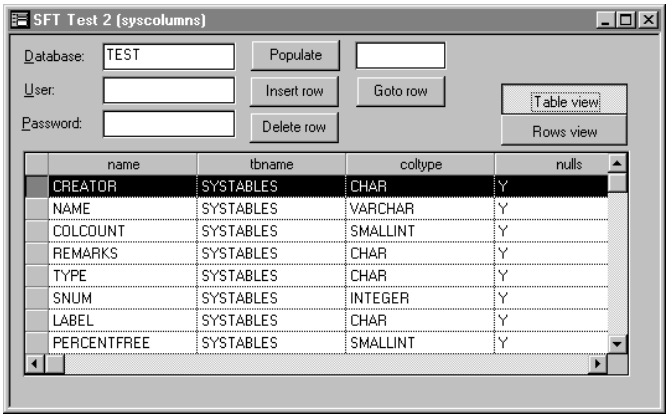

Figure 5. The second sample application showing SYSCOLUMNS in table-view.

"ODBC" Means "Overwhelming Database Confusion"

RanFlam

An update on an ODBC problem posted last month to our Web site . . .

We recently started using ODBC connectivity to Microsof SQL Server 6.5 and discovered a very confusing Centura "feature": GUPTA.INI creates unwanted "shortcuts" when connecting to an ODBC data source. Whenever an ODBC connection is established, a GUPTA.INI is created (if it didn't already exist). The location of the GUPTA.INI is specified either in WIN.INI in the section of SQLWindows, parameter [SQLWINDIR], or—if the latter isn't defined (such as in a non-R&D PC)—in the \WINDOWS directory.

In our case this file specifies the ODBC data sources that were successfully connected at some point in time.

GUPTA.INI example

```
[GUPTA ODBC DATA SOURCES]
MSPRBASE=GPOD0000

[MSPRBASE]
RemoteDBName=GPOD0000
ODBCConnectString=DSN=MSPRBASE;UID=SYSADM;APP=TrackWise
(R) - Login module;WSID=RAN;DATABASE=msprbase
```

For consequent ODBC connections to the same data source, such as upon restarting our application, a search is made in the GUPTA.INI (invoked by SqlConnect( ) and through services apparently provided by odbsal.dll or sqlodbw.dll). If the given Data Source is already specified, the SQLDriverConnect( ) will specify the database name as it appears in GUPTA.INI for the given data source name (DSN). It will *ignore* the Database Name specified in the corresponding Data Source as managed by the ODBC Administrator!

Our customer had the situation where the corresponding DSN entry in GUPTA.INI had:

```
DSN=MSPRBASE
```

and:

```
DATABASE=master
```

This was probably caused by an unauthorized first attempt connection, which didn't have access permission to the specified msprbase, and thus defaulted to the "master" database. Runtime calls to SqlConnect( ) with SqlDatabase="msprbase" were successful, yet the *actual* connection established was to the "master" database, *not* to the desired "msprbase" (as specified for this Data Source in the ODBC Administrator). Note that Microsoft SQL Server always has a "master" database, explaining why the SqlConnect( ) was successful. Alas, the application failed immediately thereafter since the "master" database didn't have the desired application tables (as created and defined in the "msprbase" database).

At first, we resolved the problem by programmatically deleting the GUPTA.INI *before* the first SqlConnect( ) attempt, thus entirely using the Data Source information as we've defined using the ODBC Administrator. We lost *only* one week of time before coming up with this workaround.

More recently I received additional information about this "feature" from Steve Couture, a Centura developer. According to Steve:

> *"In your case you have defined an ODBC data source MSPRBASE and that is what you usually use from within SQLWindows to connect to the database. Now instead of using MSPRBASE as SqlDatabase, define an entry in SQL.INI like:*
>
> ```
> [ODBCrtr]
> remotedbname=MYTEST,DSN=MSPRBASE
> ```
>
> *. . . and then connect to MYTEST from within your application. I think you will find that no GUPTA.INI file is created, and you will correctly connect to the database defined in ODBC as MSPRBASE. The only way that connection could occur is if the SAL application used the [ODBCrtr] section of SQL.INI to rectify the database name."*

We have tried Steve's suggestion and, indeed, using a pseudonym avoids the creation of the GUPTA.INI. I hope this information helps others avoid similar troubles.

Also, reader Liam C. Drew reports:

> *"We discovered a similar problem to the one described by Ran Flam, but rather more obviously disruptive. If an entry exists in GUPTA.INI for a database name, then SQLWindows attempts to use that as a connection, rather than looking in SQL.INI for a translation. Thus, when we set up an ODBC connection to "DB17" (an Oracle database) from Lotus 1-2-3, it was entered into GUPTA.INI (not quite sure how), and SQLWindows then attempted to connect to this database via ODBC rather than using the SQLORAW router.* **CP**

Ran Flam is a developer with Sparta Systems, Inc. in Port Monmouth, New Jersey.

Check Out Named Properties in CTD

Centura Tip!

Named Properties are a feature added in SQLWindows 5.0 to store user-defined, object-specific information in the outline. Although Named Properties are primarily used with Quick Objects, they can be used to store virtually any information known at design time that will be needed at run time.

In the current releases of SQLWindows and Centura Team Developer, Named Properties aren't visible at design time; you can't see them in the outline editor, for example. A few third-party tools will let you browse named properties; and in the Tomahawk release of CTD, named properties will be displayed on the Attribute Inspector.

But for CTD 1.x users though, you can examine the named properties through the outline print capabilities of the design environment. **Figure 1** shows the Page Settings dialog launched from Centura Builder's File menu.

By checking the Include Named Properties check box, named properties will be added to the print-out as shown in **Figure 2**.—***R. J. David Burke***

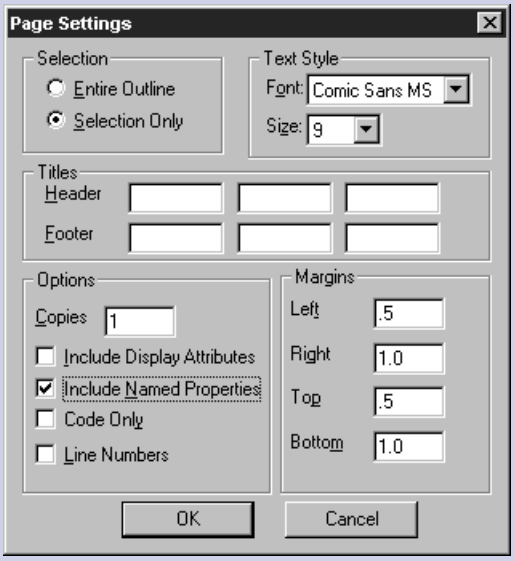

Figure 1. The Centura Builder Page Settings dialog.

```
Functional Class: OC__DVpeEvents
  Named Properties
    DISPID: (Contains not printable characters)
    LCID: (Contains not printable characters)
    LIBGUID: {F789E000-CC28-11CF-AEF7-444553540000}
    TYPEGUID: {F789E003-CC28-11CF-AEF7-444553540000}
    TYPEIDX: (Contains not printable characters)
    UFCLASS: Y
    VER: 1.0
  Description: Library for OC__DVpeEvents (UF generated class)
  Derived From
  Class Variables
  Instance Variables
  Functions
    Function: DestroyWindow
      Named Properties
        FTYPE: M
        FUNC_PROP: 0~1~24~0~
    Function: Print
      Named Properties
        FTYPE: M
        FUNC_PROP: 2~3~24~1~mode~long~3~0~0~
```

Figure 2. A sample print-out with named properties.

Access the C Runtime Library from SAL

R.J. David Burke

Hey, developers! Got 3GL jealousy? Are C programmers making fun of you because you use a 4GL? Turn their own weapons against them! R.J. David Burke shows you how to "fill in the blanks" in Centura with calls to runtime engines.

Chances are the Microsoft C Runtime Library has been placed on your machine as a deployment file for some software package you've installed. Check on your machine for a file name like MSVCRT*xx*.DLL where *xx* is a Microsoft Visual C++ major release such as 10, 20, or 40 (for Visual C++ 1.0, 2.0, or 4.0, respectively).

You can access C standard library functions by declaring them as external functions from this file. For example, sprintf is commonly used in C programs to "write" a formatted string into a string buffer. You can declare this function in SAL as shown in the following code from a Centura Team Developer application.

A sample declaration for sprintf from the C standard library.

```
♦ Library name: msvcrt40.dll
  ♦ Function: sprintf
    ◊ Description: int sprintf( char *buffer,
      const char *format [, argument] ... )
    ◊ Export Ordinal: 0
    ♦ Returns
      ◊ Number: INT
    ♦ Parameters
      ◊ Receive String: LPSTR
      ◊ String: LPCSTR
      ◊ Number: INT
```

With the above declaration, the next SAL code fragment shows an example of calling sprintf.

Calling sprintf in SAL

```
Call SalStrSetBufferLength( sBuffer, 100 )
Call sprintf( sBuffer,
  'The answer to life, the universe, and
    everything: %d', 42 )
Call SalMessageBox( sBuffer, 'sprintf result', 0 )
```

Of course, in C, sprintf is declared as accepting a variable number of arguments of any type after the first two arguments. If you wanted to use sprintf in more than one way from SAL, you'd have to declare sprintf multiple times. Each declaration would specify the same export ordinal, but the name would be different to reflect the kinds of parameters it would accept. For example:

An example of multiple declarations of sprintf to handle different parameters.

```
♦ Library name: msvcrt40.dll
  ♦ Function: sprintf_integer
    ◊ Description: sprintf_integer( sBuffer, '[…]%d[…]', n )
    ◊ Export Ordinal: 1088
    ♦ Returns
      ◊ Number: INT
    ♦ Parameters
      ◊ Receive String: LPSTR
      ◊ String: LPCSTR
      ◊ Number: INT
  ♦ Function: sprintf_float
    ◊ Description: sprintf_float( sBuffer, '[…]%f[…]', n )
    ◊ Export Ordinal: 1088
    ♦ Returns
      ◊ Number: INT
    ♦ Parameters
      ◊ Receive String: LPSTR
      ◊ String: LPCSTR
      ◊ Number: DOUBLE
  ♦ Function: sprintf_string
    ◊ Description: sprintf_string( sBuffer, '[…]%s[…]', s )
    ◊ Export Ordinal: 1088
    ♦ Returns
      ◊ Number: INT
    ♦ Parameters
      ◊ Receive String: LPSTR
      ◊ String: LPCSTR
      ◊ String: LPSTR
```

CP

R.J. David Burke works for Centura Software in between concocting contributions for this publication.

What Do *You* . . .

Continued from page 2

about performance and connectivity for Sybase, Microsoft SQL Server, and Oracle. We have run such articles in the past and we will continue to pursue them for you.

There were a huge number of individual suggestions on other topics, too. One respondent thought, "An article on SAL97 would be nice." After he typed this suggestion, he read the May issue, with its editorial on SAL97 and the Java conference report. We can't always guarantee that level of response, but we'll try.

Want to see what other readers thought about your favorite subjects? See the chart for interest levels on various editorial topics.

	Interest Level			
	High	Mid	Low	No ans.
SQLWindows	51%	26%	16%	6%
Centura	88	11	1	0
SQLBase	38	14	42	6
Product reviews	31	57	8	4
Dynalibs	35	48	14	4
Class library design and use	77	19	1	3
Centura Internet QuickObjects	38	36	21	5
QuickObject framework	18	50	26	7
Centura Web Publisher	34	45	18	3
ForeSite application server	29	45	19	8
Tomahawk/CTD 2.0	61	27	9	4
SQLWindows "Cyclone"	30	33	26	14
OLE automation	47	38	11	4
SAP	15	28	50	8
Centura Java features	55	38	6	1
User interface design	61	31	3	5
3-tier design and coding (CICS, Tuxedo, CORBA)	32	43	21	4
Using API/OS extensions	53	30	10	6
Centura application servers	41	44	11	4
Optimizing backend databases	55	30	11	3
Replication (Ranger)	19	35	41	6
Lotus Notes	14	16	63	8
Industry/vendor news	17	52	23	6
SQLBase advanced topics	37	12	45	6

I was surprised at the very high level of interest in class library design. Of course, virtually every developer uses OOP, but I thought that most of them had already settled on the techniques they liked best. Most of our current articles feature class definitions of one kind or another; but we'll also work harder to bring you smart, useful articles on general class library design in the future.

User interface design was another surprise, another area where I thought most developers had achieved a comfort level. Again, many *Centura Pro* articles feature isolated widgets and techniques that can improve an interface; but we'll now also look for higher-level articles that discuss how to coordinate the entire user interface.

Hey, where are these articles going to come from, anyway? Well, in addition to our current stable of talented authors, we also got dozens of respondents who indicated an interest in writing for *Centura Pro*. We welcome fresh voices in the newsletter, and look forward to publishing their articles. **CP**

Stepping Out with the Debugger

Centura Tip!

Although both SQLWindows and Centura Team Developer provide powerful debugging facilities, they don't provide all the bells and whistles that you find in other development environments. One thing that I would like to see is Step Out functionality to complement Step In and Step Over. A typical scenario is where I've stepped into a function (or message handler) and I suddenly realize that I don't want to be here. I should've stepped over instead of stepping in.

With a Step Out feature, the debugger would allow the execution of the rest of the function to continue and halt execution when the line following the function call is about to be executed. Step Out is likely to appear as a new feature in the Tomahawk release of CTD. But for CTD 1.x and SQLWindows, you can use this manual workaround.

First you need to access the call stack window. In CTD, use the Debug toolbar. In SQLWindows, click on the Stack button in the debug window. The top item in the call stack list is the current execution context. To step out, you want to return to the previous execution context, the second from the top item.

Click on the second from the top item in the call stack window. The SAL outline will *automagically* scroll to reveal where this call occurs in your source code. Click on the following line in the outline and then set a breakpoint there (F9). Now click on the Continue button (SQLWindows) or the Go button (CTD). Execution continues from the current point to the breakpoint you've just set.

You've effectively stepped out. When you get comfortable with this technique, you can use it to step out multiple levels, which is sometimes useful as well.

—R.J. David Burke