Clarion Magazine

Current articles: XML

Current news: XML

PDF For May, 2003

All Clarion Magazine articles for May, 2003 in PDF format.

Posted Monday, June 02, 2003

Creating A Designer Interface In Clarion (Part 1)

This three part series by David Podger and Deon Canyon examines selected code from David and Deon's MediaSim touch screen application, written entirely in Clarion. This first article provides an overview of the application's purpose and a summary of the design requirements. Part 1 of 3.

Posted Thursday, June 05, 2003

Creating A Designer Interface In Clarion (Part 2)

This three part series by David Podger and Deon Canyon examines selected code from David and Deon's MediaSim touch screen application, written entirely in Clarion. This installment covers the MouseDown event code, including important issues regarding control order in the window declaration. Part 2 of 3.

Posted Thursday, June 05, 2003

Creating An XML RSS Web Site Summary With Clarion 6 (Part 1)

Ever wonder how easy or difficult it is to work with XML? Consider the RSS specification, which is used to create summaries of items on a web site. With relatively little work, you can create RSS files in Clarion, using some of the new XML capability in Clarion 6. In this installment, David Harms looks at the RSS specification and presents some Clarion source. Part 1 of 2.

Posted Friday, June 06, 2003

The ClarionMag Third Party Product Directory

A reminder that Clarion Magazine now has a categorized third party product directory! If you're the owner/maintainer of any Clarion-related product or resource, commercial or free, you can maintain your own product listings here! Your additions/changes are posted immediately. Besides the usual

News

SelaSoft xToolTip And Clarion6

ClarionPost.com Login Problems

xmlFUSE 1.1 Includes RSS Reader

Lindersoft Server Upgrade

Fomin Report Builder v.2.86 (C6 Compatible)

Riebens System Office Closed Through Monday, July 7 2003

PD Browse Button Lookup 60-03 And PD Drops 60-02

product types, there are categories for included components and compatibility, so you can easily find, say, all C6 compatible products, or those which come with source code. The directory has its own tab, so for ready access just remember to click on Products!

Posted Saturday, June 07, 2003

Weekly PDF For June 1-7, 2003

All ClarionMag articles for June 1-7, 2003 in PDF format.

Posted Monday, June 09, 2003

Creating A Designer Interface In Clarion (Part 3)

This three part series by David Podger and Deon Canyon examines selected code from David and Deon's MediaSim touch screen application, written entirely in Clarion. This installment covers the MouseUp event code, including repositioning controls. Part 3 of 3.

Posted Wednesday, June 11, 2003

Many Reports: Many Printers

Faced with a customer request to assign specific printers to specific tasks, Henry Plotkin turns to the PrintDialog library function.

Posted Friday, June 13, 2003

Creating An XML RSS Web Site Summary With Clarion 6 (Part 2)

Ever wonder how easy or difficult it is to work with XML? Consider the RSS specification, which is used to create summaries of items on a web site. In this second of two parts, David Harms looks in detail at the code.

Posted Friday, June 13, 2003

Weekly PDF For June 8-14, 2003

All ClarionMag articles for June 8-14, 2003 in PDF format.

Posted Monday, June 16, 2003

Clarion Magazine's RSS Feeds

Wondering what those new XML icons are on the ClarionMag home page? They're RSS feeds, and in combination with an RSS reader they make it possible for you to get notification of articles and news items when they appear on the web site. No more waiting for the weekly update emails!

Posted Friday, June 20, 2003

UK Clarion Training September 2003

Updated Office Templates Documentation

EU VAT At CapeSoft

Recent CapeSoft Updates

Ezhelp 2.57

Draw 2.03 - Draw Goes Gold

Insight Graphing 1.09 beta

Insight Graphing For C6

RDraw Class To Be Released This Week

1stLogoDesign Fourth Of July Sale

gReg Fourth Of July Sale

Gitano Fourth Of July Sale

RPM For C6 EA4.5

ClarionTools Clarion 6 Support

SCA Microtemplates VAT On EU Sales Starting July 1st, 2003

LinderSoft EU-Based Customers And VAT

SetupBuilder 5.0 Beta Due Soon

SimTabTree Template 1.04

ClassViewer Web Pages Updated

SimTabTree Template Updated

Look Good Package Redesigned

Fenix: Beta 2.0 Shipped

Fomin Report Builder v.2.86

PD Browse Button Lookup Version 60-02 Released

Software Submission Service

ClassViewer 6.0 Released

SetupBuilder 4.03 Final Release Candidate

How to Display An Image In A ListBox Header (Part 1)

Have you ever wished that you could place an image on a listbox header? In this article Randy Rogers explain the first of two techniques (that is, hacks) that he uses to place little triangles in the header, like those used in Microsoft Outlook to display a column's sort order. Part 1 of 2.

Posted Monday, June 23, 2003

A Tree In A Page Loaded Browse: Update And Delete Logic

In this installment, Ronald van Raaphorst continues his discussion of a simple alternative to the RelTree template with a discussion of the ins and outs of updating and deleting records.

Posted Monday, June 23, 2003

How to Display An Image In A ListBox Header (Part 2)

Have you ever wished that you could place an image on a listbox header? In this article Randy Rogers explains how to use subclassing to place little triangles in the header, like those used in Microsoft Outlook to display a column's sort order. Part 2 of 2.

Posted Wednesday, June 25, 2003

A String To CString Converter

Sometimes you need to pass CSTRINGs to a function, and it's inconvenient to create variables for the purpose - you just want to pass in string literals. But Clarion (at least through version C6 EA4-5) all literal strings are STRINGs. So how do you do it? With a class, of course.

Posted Thursday, June 26, 2003

Creating XML Files With The Clarion 6 DOM Parser

Clarion 6 ships with class wrappers for the CenterPoint XML library, which includes SAX and DOM parsers. Although there is as yet no documentation for the Clarion class wrappers, they're actually quite easy to use. David Harms shows how to use the DOM classes to write an RSS XML file.

Posted Thursday, June 26, 2003

Newsletter Service Update

RPM Email Survey

True Edit-In-Place Template

SimTabTree Template Update

Early Bird SQL Seminar Special Ends Soon

RDRAW Demo

Accounting System Demo

New Permanent cpTracker Price

PlugIT Clarion 6 Compatible

CHT C55/C6 Co-compatible Build Coming July 31

File Manager 3 Beta 18 Released

EasyResizeAndSplit Special Offer

CPCS C6 Beta for EA4-3 Released

New SimTabTree Template Demo

EasyAutoEntry Review At ClarionShop

Beta Testers Wanted For A Clarion XML Document Reader

Search the news archive

Clarion Magazine Office Closed July 1-7, 2003

The Clarion Magazine office will be closed July 1-7, 2003. Subscription orders will be processed automatically as usual, but email will in all likelihood go unanswered until July 7.

Posted Monday, June 30, 2003

Looking for more? Check out the **site index**, or **search the back issues**. This site now contains more than 700 articles and a total of over a million words of Clarion-related information.

Clarion Magazine

Creating A Designer Interface In Clarion (Part 1)

by David Podger and Deon Canyon

Published 2003-06-05

It was one of us, Deon Canyon, who began this project. A health research worker and a super-user of computers rather than a programmer, he wanted to create touch screen applications for use in public health education projects. So, without fully realizing the implications, the other of us, David Podger, offered to write *in Clarion* an environment within which Deon could declaratively write these applications himself. This modest offer gradually took over more of David's life than he (David) had expected.

We had already designed and written a disease modeling environment together which gave us something to start with. This ran by interpreting a combination of scripts and form-based data and provided the logical framework. Its graphic display was via HTML only, courtesy of File Explorer from CapeSoft, and its only data entry interface was via fixed format option boxes and value fields. This fixed interface had to be replaced by a fully definable one controlled by a third-party designer (Deon, in the first instance).

A touch screen is a mouse emulator, so we were able to develop the app without even owning a touch screen. There were just three mouse events we had to deal with: MouseDown, MouseMove and MouseUp. How could this be so difficult?

A quick look at educational touch screen applications

An educational touch screen application asks questions, and, depending on the responses, displays different information using multi-media. So it may, among other things, play a sound file, display images or run a short movie in reaction to a particular response.

Sets of questions are normally organized into scripts. A typical touch screen app

begins with an *attractor* script displaying a succession of screens of multi-media information, endlessly looping through them until someone (a client) presses the screen. A branch to a new script then occurs, perhaps to a menu script, offering a choice of subjects. In effect this script is asking a question - 'What subject do you want to know about?' - and, depending on the response, kicks off another script to present that subject in an interactive fashion. When a script ends it may return to the menu script, or drop through into a second presentation script. These choices are in the designer's hands.

Defining the problem

It's the "fully definable" interface for the designer that is the challenge. It just has to be a graphic interface. Suppose the designer wishes to create a heading and set it in a background panel looking something like Figure 1.

Figure 1. A heading on a background

The text, its font, size and color and the color of the panel can all be defined using the fields in a conventional Clarion form. But to move the panel and its text around and re-size it using this method is truly inconvenient. Imagine having to enter the X and Y co-ordinates of a panel's top-left corner, and the width and height, as numbers! Or, only somewhat better, be forced to use the arrow keys on the keyboard (in combination with the Ctrl and Shift keys) to manipulate the panel. Compare this to creating handles with one mouse click and dragging the panel to its desired location with a quick mouse movement, or dragging on individual handles to re-size the panel, as in Figure 2.

Figure 2. Resizing/moving the panel

There is no comparison.

So we decided to implement a variation on Clarion's own graphic design interface in Clarion itself. There wasn't a choice.

The many different controls required

A touch screen needs quite a variety of controls. Further, there may be multiple instances of the same control type. At its present stage of development our product, MediaSim™, has the following controls:

1. The text for a Question and its associated background panel
2. Response texts and their associated panels (up to 10 on screen at a time)
3. Response buttons, with their associated response texts within the button or alongside
4. An Option Box containing multiple choices (an alternative to 2. and 3.)
5. Navigation buttons (up to six on screen at a time), such as Back and Next
6. A primary image that can occupy the whole screen
7. Up to four side-panels images (Top, Bottom, Left and Right)
8. A File Explorer area for Html or PDF pages (and hopefully PowerPoint)
9. A movie area
10. A tool box (called "control-box" in this article)

The designer may define any of five characteristics of the above controls as appropriate, including:

1. Size
2. Location
3. Font
4. Color
5. Embedded image(s)

For instance, each of the Response and Navigation buttons has up to four GIF's, representing their normal, roll-over, pressed and disabled states. The SimSoft templates from South African software supplier Simplified Software allow button images to be loaded from hard disk at run time, and this feature, in turn, allows our designer a choice of images. Similarly, solid.software's ImageEx accessory allows a primary image and four panel images to be loaded rapidly at run time

In addition to the above controls, the design environment must have control labels so the designer can recognize each instance of a control. These "Tiny Controls" are small prompts with a colored background located in the top left corner of any control needing them (buttons for example are numbered 1 to 10). They move around in synch with their parent controls. Lastly, there are the handles, very small colored prompts (red for the currently selected control, or blue for multiple selected controls other than the last-selected) that become

visible on a control or set of controls when selected by the designer.

A non-programmer needs to see the above controls *as they will appear in the finished application*. Just having outlines, as in Clarion's own design environment, is not sufficient. Consider, for instance, the option box shown in Figure 3.

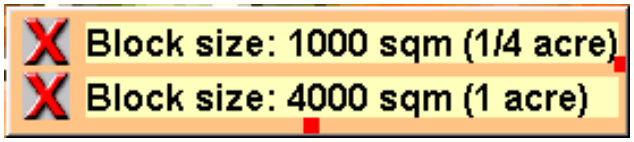

Figure 3. An option box

Making it larger should smoothly increase the controls within it, so the designer can immediately see (as in Figure 4) that the box must be made wider:

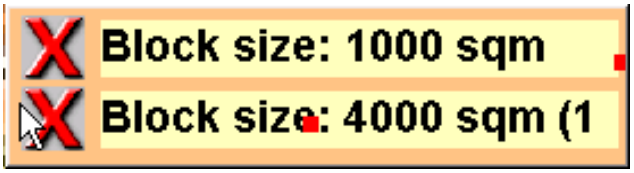

Figure 4. A resized option box that is too narrow

There are limits, however, to what can be shown at design time. The Movie and HTML areas are only shown in outline, but can be seen in living color in their specified location and size at the click of a button. All other controls are shown in immediate WYSIWYG.

It follows that the designer window must be full-size, with no appframe banner, no menu and no toolbar. This apparently simple requirement is difficult to satisfy, since the designer window is several levels deep within a MDI application.

And, if the designer window is full-size, the designer's tool box must be floating and re-sizable so that it can be pushed out of the way. Because this is not a Clarion "toolbox" the box, from now on, will be called the "control-box". The many types of controls needed in the control-box were a challenge. The solution: treat the box as another resizable, relocatable control in the window, which is why it is one of the ten controls listed above. But this is a non-trivial hand-coding job, so why use Clarion?

Why use Clarion?

Writing a subset of Clarion's own design interface in Clarion (and for some features, a superset) in fact turned out not to be seriously difficult. The code is a tad clumsier than it would have been in C, but it is still quite readable. Add to this the third-party tools mentioned above that do so much for a modest cost, and then add the speed with which the many normal browses and forms could be written, and the balance swings strongly in Clarion's favor.

The implementation steps

The first thing to decide in implementing this kind of interface is a question of basic methodology. How can Clarion's mouse events (`MouseDown`, `MouseMove` and `MouseUp`) best be used to create an easy-to-program graphic design environment? Mouse events can only be detected within a defined `REGION`, but how many regions are needed? Some brief experiments showed that just one is sufficient and that it should cover the whole screen. Using this method, the `MOUSEX` and `MOUSEY` functions can return a value, no matter where the mouse is clicked. They won't always do so, as will be seen later, but they *can*.

The implementation steps are dictated by the user's (that is, the script designer's) actions. Step-by-step, this is what happens:

Step 1 – The user presses on the left mouse button. Because this occurs within a region, it generates a control event. So, under Control Events, code can be embedded for **Accepted (MouseDown)**, and the mouse's position can be found. Early on in the `MouseDown` code two values are obtained that define where the mouse was clicked:

```
MX = MOUSEX()
MY = MOUSEY()
```

Knowing the position of a `MouseDown` event, the next step is to decide which control was just selected, and specifically which part of that control. The mouse's position somewhere inside the whole screen is converted into the identity of the selected control (details later). Apart from the null case where the mouse is clicked in background space between the controls, there are three possibilities:

1. A new control is selected, not one of the borders of an already selected control
2. A red-handled control exists and one of its borders is selected
3. A chain of blue-handled controls exists and an additional one is selected

Code must be written to handle the above possibilities.

Step 2 – The user keeps the left button down and moves the mouse. Under **Control Events**, code can be embedded for `MouseMove` to detect each new position the mouse is moved to. The user is either moving a whole control or dragging one of its borders. The code for this step must be brief and efficient. Therefore, only an outline of the control (or a selected border) is displayed while the control is being moved. If the user clicks down and up in the same place, this step is skipped.

Step 3 – The user lets go of the left button. Under Control Events, code can be embedded for `MouseUp` to detect where the mouse is when the button is lifted. The movement of the outline control (if any) is completed and the control itself is displayed at its new position and with its new size, and the variables holding these values are updated.

That's the outline – next, it's time for the details. Part 2 will cover Step 1, Part 3 will cover Steps 2 and 3.

David Podger has been an independent Clarion developer in Australia for a decade. He presently lives in Katherine in the Northern Territory, where he sells a specialised accounting application for remote communities.

Dr Deon Canyon is a Lecturer (translation: Assistant Professor) in public health and medical entomology at James Cook University in North Queensland, Australia. His current research interests include: Disease control simulation models for use in distance education; Lymphatic filariasis (a disease caused by thread-like parasitic worms) and the Aedes polynesiensis mosquito in transmission and control of the Pacific Head Lice (Pediculus capitis); Health in remote settlements and the use of touch screen kiosks to improve it; Aedes aegypti mosquito behavior, physiology and ecology . Deon is 38, married, and has three children. He likes rock climbing but busted his knee and can't do it anymore.

Reader Comments

Add a comment

Hi Dave - nice articles, but where is the sample code...
Val, The sample code is what's contained in the text...

Clarion Magazine

Creating A Designer Interface In Clarion (Part 2)

by David Podger and Deon Canyon

Published 2003-06-05

In Part 1 we introduced the concept of a touch screen application development environment for use in public health education projects, and showed how our development focused on just three events, `MouseDown`, `MouseMove`, and `MouseUp`. This part covers the `MouseDown` code, and Part 3 will cover the `MouseMove` and `MouseUp` events.

The first question is, of course, is just which control out of all those crowding the screen was actually clicked?

Which control was just selected?

Under **Control Events**, code can be embedded for **Accepted (MouseDown)**, for `MouseMove` and `MouseUp`. Early on in the `MouseDown` code two values are obtained:

```
MX = MOUSEX()
MY = MOUSEY()
```

Together, the `MOUSEX()` and `MOUSEY()` functions give the X,Y position within the whole-screen `REGION` where the mouse was clicked. But the question is, which particular control was selected? The controls are constantly changing position and changing size, so they are moving targets. To keep track of them a queue (`ControlQ`) is initialized with the positional details of every control when the window opens, and is kept up-to-date throughout. Whenever a `MouseDown` event occurs the queue is searched to see which control the event refers to. The code is:

```
SORT(ControlQ,CQ:Area,CQ:Control,CQ:Index)
LOOP Q# = 1 TO RECORDS(ControlQ)
```

```
GET(ControlQ,Q#)
IF  MX >= CQ:CX AND MX < (CQ:CX + CQ:CW)    AND |
    MY >= CQ:CY AND MY < (CQ:CY + CQ:CH)
    ! clicked control located, do something about it
```

The `ControlQ` has a prefix `CQ`, so the variables `CQ:CX`,`CQ:CY`,`CQ:CW` and `CQ:CH` give the X,Y position of the top left corner of the control and its width and height. The above code finds the control where the `MouseDown` occurred. The search loops serially through `ControlQ`, in ascending order by control area. Searching in this order finds the smallest controls first, even if they are positioned over larger ones. The control-box control and its various groups of controls are given arbitrary areas (1 and 2, respectively) so they can be recognized and skipped over if needed.

By using a queue to pinpoint a selected control, all the code for a `MouseDown` event can be general. It will work no matter which control is detected, if written correctly. This is true also for the coding of `MouseMove`. Only the `MouseUp` code must be made particular to each control type. We'll go into more detail about the code, but first the definition of the designer's window needs some explanation. The effect of the order of control definitions within a window

Suppose a `REGION` covering the entire window is defined at the very beginning of a window's source definition. Placing it above (before) the window's other active controls (i.e. those within the control-box) has the effect of making them *inoperative* as controls. You click on any control and nothing appears to happen! To allow a control to respond to mouse clicks, it must be *above* the `REGION` in the window declaration. After moving the region definition lower and lower, we ended up placing it last, under all the controls.

As a result of this positioning, a mouse click on any enabled control (such as a button within the control-box) triggers events for that control. Mouse activity anywhere else is detected by the `REGION` and reported as one of the three mouse events. A click on a disabled control, such as a disabled button, drops through to become a mouse event detected by the underlying region. As well, a disabled button is always visible, no matter where it is defined, and is easily colored at run time. This made it most suitable for the control-box, which must always stay on top.

The order of definition is also respected. When the window is first opened, buttons lower down in the window's definition appear above earlier defined buttons no matter if they are disabled or not. This is because buttons are created on the window in the order they are declared.

Ordinary boxes are also useful. They may have rounded corners and are readily colored at run time. Boxes lower down in the window's definition appear above earlier defined boxes. They are efficiently rendered on screen. As a result, they are suitable for all outline movements. The background panels for questions and prompts are boxes. At first we made them disabled buttons, but these hid the movement of boxes (the gray outlines boxes disappear under them).

Prompts also get precedence in the visibility war. This is why handles are prompts. So are the "Tiny Controls" which identify each button by number. A careful observer would see that outline boxes slide underneath them, but they are so small that it doesn't matter.

The ControlQ Queue

As noted above, the ControlQ is central to deciding which control has been selected in a MouseDown event. It fulfills other purposes as well. Here is its definition:

```
ControlQ    QUEUE, PRE(CQ)
Control        LONG      ! control's serial number
CX             LONG      ! current X,Y position
CY             LONG
CW             LONG      ! current width and height
CH             LONG
WhichMove      STRING(8) ! eg. Html, Movie, Question
Index          BYTE      ! instance of a control
ChangeControl  LONG      ! CHANGE(Field Control,1) ticks control-box
BeginHandle    LONG      ! Field Control or 0 = not
EndHandle      LONG      !    "     "
MovementBox    LONG      !     "     "
Area           LONG      ! width * height
TinyControl    LONG      ! identifies tiny controls
HiddenFlag     BYTE      ! marks control as hidden
            END
```

`CX`,`CY`,`CW` and `CH` together define the location and size of a control; `WhichMove` is the control name, and identifies control-specific code to use; `Index` is a serial number for controls with multiple instances

The next four variables make it possible for `MouseDown` and `MouseMove` code to be generic. `ChangeControl` defines which control-box control is to be ticked when an in-window control is selected. The value in this variable is initialized with a `FieldControl` number when the designer's window opens, and does not need any subsequent change.

The next three variables (`BeginHandle`,`EndHandle`,`MovementBox`)all

begin as zero. They also play a part in the generality of the `MouseDown` and `MouseMove` code. They are the Field Control numbers for the first and last handles in a chain of linked controls and for any movement box associated with a control. A movement box provides the gray outline of a control when it is moved using `MouseMove`.

`Area` is used to sort `ControlQ` into ascending order for the search to find which control has been selected when a mouse event occurs. The controls `TinyControl` and `HiddenFlag` are experimental only at present.

When a control is selected it displays eight handles. Each handle marks a border. The eight values for the variable `Border` are:

- 'Bottom'
- 'Right'
- 'Top'
- 'Left'
- 'BottomRight'
- 'TopRight'
- 'BottomLeft'
- 'TopLeft'

If the Border variable is blank, this means that the control as a whole has been selected and not one of its eight handles.

The MouseDown Code

The `MouseDown` code is placed at the embed point **Accepted for Region1, after the Generated Code**, and has to deal with four possibilities:

- A new control is selected, not one of the borders of an already selected control
- A red-handled control exists and one of its borders is selected
- A chain of blue-handled controls exists and an additional one is selected
- The background space is selected (i.e. mouse down where there is no control)

The code for the first possibility follows. It has four purposes: firstly, to clear handle and movement box pointers from `ControlQ`; secondly, to update the control-box (via a ticked check box) and show which control is now selected (and its Index number if needed); thirdly, to display red handles and display a movement box of the same size and in the same location as the selected control;

and fourthly, to restore everything to a default state if background space has been clicked:

```
...
MX = MOUSEX()
MY = MOUSEY()
! clear saved positions
SaveX = 0
SaveY = 0
!   if no border (eg. 'TopLeft'), work on whole control
IF Border = ''
  SORT(ControlQ,CQ:Area,CQ:Control,CQ:Index)
  ! find a control
  LOOP Q# = 1 TO RECORDS(ControlQ)
    GET(ControlQ,Q#)
    IF MX >= CQ:CX AND MX < (CQ:CX + CQ:CW) AND |
       MY >= CQ:CY AND MY < (CQ:CY + CQ:CH)
      IF CQ:Area = 1 AND AllowOrdering
        ! skip control-box
        CYCLE
      END
      IF CQ:BeginHandle
        ! remove all handles and movement box
        LOOP P# = CQ:BeginHandle TO CQ:EndHandle
          DESTROY(P#)
        END
        CQ:BeginHandle = 0
        CQ:EndHandle = 0
        IF CQ:MovementBox
          DESTROY(CQ:MovementBox)
          CQ:MovementBox = 0
        END
        PUT(ControlQ)
      END
      IF CQ:ChangeControl
        ! check box present in tool-box, so
        ! untick all other check boxes in  DISPLAY
        ! control -box and tick this one
        DO ClearAllMovements
        CHANGE(CQ:ChangeControl,1)
      END
      IF CQ:Index
        ButtonTarget = CQ:Index
        DISPLAY(?ButtonTarget)
      END
      ! display the primary movement box
      XBox = CQ:CX
      YBox = CQ:CY
      WBox = CQ:CW
      HBox = CQ:CH
      DO SetRedHandles
      IF ControlChain
        AreaOrder = TRUE
        ! show chained movement box(es)
        DO ShowBoxes
```

```
      END
      SaveX = MX
      SaveY = MY
      BREAK
    END
  END
  ! restore to control order
  SORT(ControlQ,CQ:Control,CQ:Index)
  IF SaveX                      AND |
     ~(SaveControl = CQ:Control AND |
       SaveIndex   = CQ:Index)
    ! found a control, but
    ! different from current one
    ! save the new control and
    ! its index (may be zero)
    SaveControl = CQ:Control
    SaveIndex   = CQ:Index
    IF ~ControlDown AND ControlChain
      DO DestroyBlueHandles
    END
  ELSIF ~SaveX
    ! clicked background space
    IF ControlChain
      DO HideBoxes
      DO DestroyBlueHandles
    END
    DO ClearAllMovements
    HIDE(?H1,?H8)
    ButtonTarget = 1
    DISPLAY
  END
ELSE
...
```

The routines called from within the above code do the following:

- `ClearAllMovements` sets all check boxes in control-box to zero (unticked).
- `SetRedHandles` sets the red handles appropriate for each CQ:WhichMove.
- `ShowBoxes` creates, positions, colors and unhides secondary movement boxes.
- `HideBoxes` destroys movement boxes as part of a general clean-up and reset.
- `DestroyBlueHandles` does exactly what its name suggests.
- The variable `ControlChain` appears in the above code. Its value is set by code that *precedes* the code shown. See below for where it is set.

Continuing with the code for the case where a red-handled control already exists and one of its borders is selected:

```
IF MX >= CQ:CX AND MX < (CQ:CX + CQ:CW) AND |
  MY >= CQ:CY AND MY < (CQ:CY + CQ:CH)
  SaveControl = CQ:Control   ! save current control
  SaveIndex   = CQ:Index
  XBox = CQ:CX
  YBox = CQ:CY
  WBox = CQ:CW
  HBox = CQ:CH
  IF ControlChain      ! a blue-handled chain exists
    AreaOrder = FALSE
    DO ShowBoxes       ! show chained movement box(es)
  END
  SaveX = MX
  SaveY = MY
ELSE              ! clicked outside current control
  DO DestroyBlueHandles
END
```

The routines called by the above code have already been described.

Where a chain of blue-handled controls is being built and an additional one is selected, the required code actually *precedes* all the code shown above. To add another control to a chain, the ControlKey must be depressed when the MouseDown event occurs and a red-handled control must already exist:

```
IF BAND(KEYSTATE(),0200h) AND ?H1{PROP:HIDE} = 0
  ! this makes current control blue and next one red
  ControlDown = 1
ELSE
  ControlDown = 0
END
IF ControlDown
  LOOP P# = 1 TO 8
    IF P# = 1
      GET(ControlQ,CQ:Control,CQ:Index)      ! collect first and last
      CQ:BeginHandle = CREATE(0,CREATE:Prompt)
      PUT(ControlQ,CQ:Control,CQ:Index)             ! update ControlQ
    ELSIF P# = 8
      GET(ControlQ,CQ:Control,CQ:Index)
      CQ:EndHandle = CREATE(0,CREATE:Prompt)
      PUT(ControlQ,CQ:Control,CQ:Index)
    ELSE
      X# = CREATE(0,CREATE:Prompt)      ! create the new blue handles
    END
  END
  C# = 0                     ! position, colour and show blue handles
  LOOP P# = CQ:BeginHandle TO CQ:EndHandle
    C# += 1
    CASE C#
    OF 1
      GETPOSITION(?H1,PX,PY)
    OF 2
      GETPOSITION(?H2,PX,PY)
```

```
    OF 3
      GETPOSITION(?H3,PX,PY)
    OF 4
      GETPOSITION(?H4,PX,PY)
    OF 5
      GETPOSITION(?H5,PX,PY)
    OF 6
      GETPOSITION(?H6,PX,PY)
    OF 7
      GETPOSITION(?H7,PX,PY)
    OF 8
      GETPOSITION(?H8,PX,PY)
    .
    SETPOSITION(P#,PX,PY,4,4)
    P#{PROP:FillColor} = COLOR:Blue
    UNHIDE(P#)
  END
  ControlChain = TRUE             ! a control chain exists
END
```

We have now described the code for the `MouseDown` event. The code is actually written in the following order, but discussed out of sequence to (hopefully) make it easier to follow:

1. A chain of blue-handled controls is being built and a new one is selected
2. A brand new control is selected
3. The background space is selected (i.e. mouse down where there is no control)
4. A red-handled control exists and one of its borders is selected

In the next article we will explain the code for the `MouseMove` and `MouseUp` events.

David Podger has been an independent Clarion developer in Australia for a decade. He presently lives in Katherine in the Northern Territory, where he sells a specialised accounting application for remote communities.

Dr Deon Canyon is a Lecturer (translation: Assistant Professor) in public health and medical entomology at James Cook University in North Queensland, Australia. His current research interests include: Disease control simulation models for use in distance education; Lymphatic filariasis (a disease caused by thread-like parasitic worms) and the Aedes polynesiensis mosquito in transmission and control of the Pacific Head Lice (Pediculus capitis); Health in remote settlements and the use of touch screen kiosks to improve it; Aedes aegypti mosquito behavior, physiology and ecology . Deon is 38, married, and has three children. He likes rock climbing but busted his knee and can't do it anymore.

Reader Comments

Add a comment

Clarion Magazine

Creating An XML RSS Web Site Summary With Clarion 6 (Part 1)

by David Harms

Published 2003-06-06

Last month I wrote an introductory article on XML, describing (briefly) what XML is, and what tools you are likely to need. One reader, Brian Staff, left this comment: "Readers do need to realize that this XML arena appears quite simple at first, but it quickly becomes a deep complex world." Brian is exactly right – at first, XML seems almost too simple to be of much use. It is, after all, just a semi-standardized way of presenting data using tags (text surrounded by < and >) to describe that data. While even the simplest XML tasks sometimes take unexpected dips and dives,however, the shallow end of the pool really is quite inviting.

By way of example, I'm going to spend this week and next showing how to create a XML document known as an RSS file (or feed). This is an XML format used (typically) to describe headlines and abstracts on web sites. RSS files can be created by hand (i.e. notepad.exe), with one of various utility programs, or on the fly by the web server software.

Since Clarion 5.x doesn't have any native XML capability I'll be using Clarion 6, but the code is quite easy to follow even if you're not part of the early access program and have never seen C6.

Who/what uses RSS?

Most often, RSS files are created by web servers or web site management software, and read by utility programs called aggregators. Aggregators can be server-based programs that harvest summaries from other sites, and distill the results for analysis and/or presentation (for instance, Feedster is a search engine that looks only at RSS feeds).

These days personal aggregators are all the rage. If you don't yet have an RSS aggregator, chances are you will soon. With an aggregator you can keep tabs on numerous web sites without having to visit them all yourself – all that's necessary is for the web site to supply an updated RSS file (and yes, a ClarionMag RSS feed is coming soon).

There is also a huge synergy between web logs such as Blogger and RSS (interestingly, Google recently purchased Pyra Labs, makers of Blogger). Web logs (more commonly called *blogs*) are basically a standardized way of presenting an online diary. And if you have, say, a dozen blogs you like to read (friends, family, movers and shakers in your industry, etc) you'd probably start out visiting those web sites on a regular basis, on the chance someone's posted something new. Enter the aggregator. If those blogs supply RSS feeds, as many now do, your aggregator can poll the web sites for you and let you know when something has changed by presenting you with a title and summary of the new item. At that point it's up to you whether you want to go to the web site to see the rest (or have your aggregator pull down the page and show it to you in its own browser window).

The uses for RSS feeds are many and varied. At least one Clarion developer I know, Mark Riffey, is considering building an RSS aggregator into his applications so he can send weekly tips directly to his apps, rather than via email.

Figure 1 shows a screen shot of a NewzCrawler, one of the more popular RSS aggregators (NewzCrawler also functions as a newsgroup reader and browser, and has some blogging capabilities.)

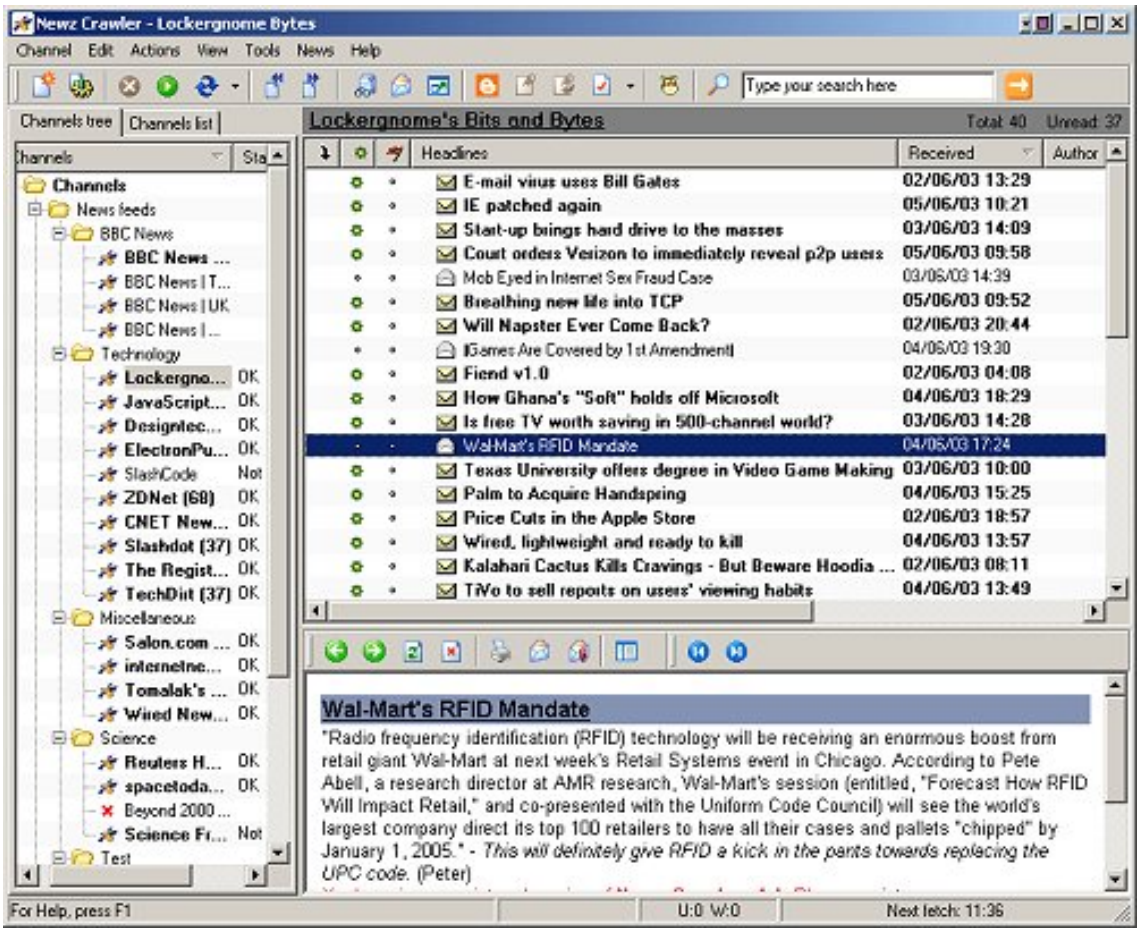

Figure 1. The NewzCrawler RSS aggregator

In Figure 1 you can see a list of RSS "channels" NewzCrawler is tracking, along with current headlines and associated summaries. Each channel corresponds to an RSS file supplied, somewhere, by a (web) server. Here's an abbreviated version of the sample RSS file from the UserLand web site (home of Radio Userland, a popular blog tool):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="0.91">
  <channel>
    <title>WriteTheWeb</title>
    <link>http://writetheweb.com</link>
    <description>News for web users that write back</description>
    <language>en-us</language>
    <copyright>Copyright 2000, WriteTheWeb team.</copyright>
    <managingEditor>editor@writetheweb.com</managingEditor>
    <webMaster>webmaster@writetheweb.com</webMaster>
    <image>
      <title>WriteTheWeb</title>
      <url>http://writetheweb.com/images/mynetscape88.gif</url>
      <link>http://writetheweb.com</link>
      <width>88</width>
      <height>31</height>
      <description>News for web users that write back</description>
      </image>
```

```
    <item>
      <title>Giving the world a pluggable Gnutella</title>
      <link>http://writetheweb.com/read.php?item=24</link>
      <description>WorldOS is a framework on which to build
      programs that work like Freenet or Gnutella -allowing
      distributed applications using peer-to-peer routing.</description>
      </item>
    <item>
      <title>Syndication discussions hot up</title>
      <link>http://writetheweb.com/read.php?item=23</link>
      <description>After a period of dormancy, the Syndication
      mailing list has become active again, with contributions
      from leaders in traditional media and Web syndication.</description>
      </item>
    <item>
    </channel>
  </rss>
```

Note this line near the top of the file:

```
<rss version="0.91">
```

One of the consequences of XML's flexibility is that anyone can create a specification for a given file format. In the case of RSS, this means that you have to decide which of the available specifications you will follow. 0.91 was the first widely-accepted version of RSS, and is an old format but still a safe bet. Later versions include 0.92, 1.0 and 2.0. I've gone with 0.91 in part because it's the simplest of the available formats and therefore the easiest example to tackle.

Writing the code

So how *do* you create this same XML file in Clarion 6? There are a number of possibilities. One is to use (or attempt to use) a report with the **Report to XML Global Extension** - this will convert your report to a corresponding XML file. The problem with this approach is that the RSS format isn't quite as straightforward as a typical report. There's probably a way to do it, but it looks like a lot of work.

Another approach is to look at the C6 wrapper for the Centerpoint XML parser. There really isn't any documentation to be had as of this writing, and the source looks pretty complex!

Happily, there is an easier way. If you look at the source generated for an XML-enabled report, you'll see that it uses a new class called `XMLGenerator` to create the XML file. `XMLGenerator` is all Clarion source, so even in the absence of documentation it's possible to work out what the public methods do.

And if the report can use `XMLGenerator`, so can you!

Here's some code that uses the `XMLGenerator` class to recreate the above file:

```
rss.Init('rssTest.xml')
    rss.OpenDocument('rss')
    rss.XMLVersion = '1.0'
    rss.AddXMLHeaderAttribute('encoding','ISO-8859-1')
    rss.AddTag('channel','')
    rss.AddTag('title','WriteTheWeb','channel')
    rss.AddTag('link','http://www.xmlhack.com',|
       'channel')
    rss.AddTag('description','News for web users that write back',|
       'channel')
    rss.AddTag('language','en-us','channel')
    rss.AddTag('copyright','Copyright 1999-2001, xmlhack team.',|
       'channel')
    rss.AddTag('managingEditor','editor@writetheweb.com','channel')
    rss.AddTag('webMaster','webmaster@writetheweb.com','channel')
    rss.AddTag('image','','channel')
    rss.AddTag('title','WriteTheWeb','image')
    rss.AddTag('url','http://writetheweb.com/images/mynetscape88.gif',|
       'image')
    rss.AddTag('link','http://writetheweb.com','image')
    rss.AddTag('width','88','image')
    rss.AddTag('height','31','image')
    rss.AddTag('description','News for web users that write back',|
       'image')
    rss.AddTag('item','','channel')
    rss.AddTag('title','Giving the world a pluggable Gnutella',|
       'item')
    rss.AddTag('link','http://writetheweb.com/read.php?item=24',|
       'item')
    rss.AddTag('description','WorldOS is a framework on which to ' &
      'build programs that work like Freenet or Gnutella -allowing ' &
      'distributed applications using peer-to-peer routing.','item')
    rss.AddTag('title','Syndication discussions hot up','item')
    rss.AddTag('link','http://writetheweb.com/read.php?item=23','item')
    rss.AddTag('description','After a period of dormancy, the ' &
      'Syndication mailing list has become active again, with ' &
      'contributions from leaders in traditional media and Web ' &
      'syndication.','item')
    rss.CloseDocument()
    rss.AddTag('title','Syndication discussions hot up','item')
    rss.AddTag('link','http://writetheweb.com/read.php?item=23','item')
    rss.AddTag('description','After a period of dormancy, the ' &
      'Syndication mailing list has become active again, with ' &
      'contributions from leaders in traditional  media and ' &
      'Web syndication.','item')
    rss.CloseDocument()
```

And here's the XML generated by that code:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
```

```
<rss>
 <channel>
  <title>
WriteTheWeb
  </title>
  <link>
http://www.xmlhack.com
  </link>
  <description>
News for web users that write back
  </description>
  <language>
en-us
  </language>
  <copyright>
Copyright 1999-2001, xmlhack team.
  </copyright>
  <managingEditor>
editor@writetheweb.com
  </managingEditor>
  <webMaster>
webmaster@writetheweb.com
  </webMaster>
  <image>
   <title>
WriteTheWeb
   </title>
   <url>
http://writetheweb.com/images/mynetscape88.gif
   </url>
   <link>
http://writetheweb.com
   </link>
   <width>
88
   </width>
   <height>
31
   </height>
   <description>
News for web users that write back
   </description>
  </image>
  <item>
   <title>
Giving the world a pluggable Gnutella
   </title>
   <link>
http://writetheweb.com/read.php?item=24
   </link>
   <description>
WorldOS is a framework on which to build programs that
work like Freenet or Gnutella -allowing distributed
applications using peer-to-peer routing.
   </description>
   <title>
```

```
Syndication discussions hot up
   </title>
   <link>
http://writetheweb.com/read.php?item=23
   </link>
   <description>
After a period of dormancy, the Syndication mailing list
has become active again, with contributions from leaders
in traditional media and Web syndication.
   </description>
  </item>
 </channel>
</rss>
```

You'll notice one obvious difference – Clarion adds line breaks before and after the tags. And there's one subtler difference – the `<rss>` tag is missing the `version` attribute. The former won't make any difference, but the latter certainly will. Of course you also need to declare an `XMLGenerator` object, and most likely you'll have to include the class definition so the app will compile.

XML looks pretty easy, doesn't it? You might think it would be less trouble creating this file with the ASCII driver. XML *can* get a lot more complicated than RSS 0.91, however, and even when you are creating simple XML files the Clarion class provides some benefits that might not be immediately obvious. Next week I'll provide an overview of the `XMLGenerator` class, explain in detail how the above code works, and I'll also show how to fix the problem of the missing version attribute.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of Developing Clarion for Windows Applications, published by SAMS (1995). His most recent book is JSP, Servlets, and MySQL, published by HungryMinds Inc. (2001).

Reader Comments

Add a comment

Hi Dave, I would appreciate it if you could write an...
I understand that not everyone is in the C6 EA program....
I just bought XMLfuse and it is an excellent alternative...

Clarion Magazine

Clarion News

SelaSoft xToolTip And Clarion6

SealSoft's xToolTip 1.0 has been verified as Clarion 6 compatible.

Posted Monday, June 30, 2003

ClarionPost.com Login Problems

Kelvin Chua reports that the PHP scripts for ClarionPost new registration and login are not working properly. All other functions within the portal are working accordingly. Kelvin is working on the problem.

Posted Monday, June 30, 2003

xmlFUSE 1.1 Includes RSS Reader

xmlFUSE 1.1 is now available. This new version includes improvements to the xmlFUSE helper functions for parsing XML documents, an example RSS (Really Simple Syndication) feed newsreader, a handcoded example, and additional documentation. The demo (including some source code and full documentation) is freely available.

Posted Monday, June 30, 2003

Lindersoft Server Upgrade

Due to the enormous increase in number of page requests (500 megabytes average data transferred per day), LinderSoft is moving from two shared servers to one dedicated server. The server will be up and down over the next little while as the configuration is updated.

Posted Monday, June 30, 2003

Fomin Report Builder v.2.86 (C6 Compatible)

A new Fomin Report Builder release is available. It is a free update as before. Registered users can get appropriate download information from the Live Update service at www.FominTools.com web site. This product release is compatible with Clarion 4, 5, 5.5 and 6.

Posted Monday, June 30, 2003

Riebens System Office Closed Through Monday, July 7 2003

The offices of Riebens systems will be closed until Monday, July 7 2003. All deliverables will also be posponed by a week and no email contact will be established.

Posted Monday, June 30, 2003

PD Browse Button Lookup 60-03 And PD Drops 60-02

The following updates have been posted to the ProDomus web site. PD Browse Button Lookup Class Library Beta (Version 60-03 for C55 and Clarion 6) has both some of new features and fixes to the class library. New features include: Paste support (previously, pastes were rejected); A global filter option to double or not double single quotes in filter strings (this provides compatibility with prior versions which did not automatically double quotes); Fixes relate to allowing entries without matching lookup records, the readonly disabling, and disabling an entry on update, and disabling lookups on an update. PD File Drop and Drop Combo (Version 60-02 for C6) contains changes to the internal library related to C6 and minor feature improvements.

Posted Monday, June 30, 2003

UK Clarion Training September 2003

Windowbox Software Limited has arranged for Russ Eggen to provide four days of Clarion training in the UK. The training is to be held in the beautiful surrounding of Esseborne Manor (www.essebornemanor.com) in Hampshire. The curriculum is not completed yet; however it will probably be as follows: 1) OOP; 2) Coding template wrappers for any class we may code; 3) Threading; 4) COM; 5) Debugger (subtitle, "Yes it really is good for something!"); 6) Clarion 6 issues like migration of existing applications and others; 7) Any topics the students would like covered. Places will be limited to a maximum number of 15 students, and will be on a first come first served basis. A price for the course has not been finalized, however it is expected to be between £600 and £700 per person. Included in the final price will be the four days training, all course materials, a copy of Russell's new book "Programming Objects in Clarion", all coffee breaks, and lunch. Accommodation is not included. Students will be expected to bring their own Notebook PC, with Clarion 5.5 and/or Clarion 6 installed. Email Mark Sarson to be placed in the priority list. Priority List Members will receive the final announcement two days prior to it being posted anywhere else.

Posted Friday, June 27, 2003

Updated Office Templates Documentation

Updated Office Templates documentation is now available for download from the SoftMasters web site.

Posted Thursday, June 26, 2003

EU VAT At CapeSoft

From 1 July 2003, all folk living in the EU will be required to pay VAT on Internet sales (Even if the company you are purchasing from is not in the EU). Of course, if you are registered for VAT then this does not really make any difference to you. Otherwise, if you've been saving up your pennies to buy the latest and greatest CapeSoft product, now is the time.

Posted Thursday, June 26, 2003

Recent CapeSoft Updates

Recent Updates to CapeSoft products include: File Manager 3 - v3.0 beta 19 - 24 June 2003; File Explorer 3.10 - v3.10 beta - 20 June 2003; CapeSoft Office Inside - v1.09 beta - 20 June 2003; File Manager 2 - v2.98 - 18 June 2003; NetTalk - v2.76 beta - 13 June 2003; Replicate - v1.17 beta - 5 June 2003; CapeSoft MessageBox - v1.6l - 27 May 2003; HyperActive - v1.7i - 11 Apr 2003; CapeSoft Mailer - v2.06 - 30 May 2003 (source code on sale too); Office Messenger - v2.08 - 20 May 2003 (source code on sale too); CapeSoft Email Server - v1.53 beta - 19 May 2003 (source code on sale too).

Posted Thursday, June 26, 2003

Ezhelp 2.57

Ezhelp allows you (or someone else) to add help tips to your application at run-time. Improvements in 2.57 include:The close button, which makes it clearer for users how to close the help tip.; Added CHM help file support; New documentation about how to add buttons, bullets, hyperlinks, horizontal lines and formatting options; Added the Repeat Last Format option; Fixed bugs with the F1 key, flickering tips and question mark problems; Updated examples. Tip : If you are using WinEvent in your application, then you will need WinEvent 2.96. Price: $149, free upgrade for registered users.

Posted Thursday, June 26, 2003

Draw 2.03 - Draw Goes Gold

CapeSoft Draw is the CapeSoft drawing engine which allows you to draw directly to an image control. Not only is this fast, but it has the added advantage that if the control is refreshed, it simply redraws the image, rather than re-calling each of drawing commands. Draw supports a wide range of features, some of which include: Flicker-free, very fast redraws; Images can be saved in either BMP or PNG formats; Vertical text support; Functions for creating 2D and 3D

simple and complex objects, shading, layers, transparency, real-time animation & 3D calculation functions.; Individual pixels control Draw 2 introduces the concept of layers, which allows multiple drawing surfaces, with full support for both single color transparency and soft alpha transparency. On special at $59 until 30 June 2003 when it becomes $99. Free upgrade for registered users.

Posted Thursday, June 26, 2003

Insight Graphing 1.09 beta

The long-awaited Insight update has been released and includes the following major new features: Mixed graph types (i.e. mixing lines & bars and so on); Scattergram graphs; Improved (faster and cleaner) drawing techniques; Shaded, and textured, backgrounds; Patterns (cross-hatching etc) for Black and White graphs. There are also a large number of bug fixes included, and the usual slew of template improvements, documentation updates, and minor new features. Beta price is $199 (gold price will be $299). Free upgrade for registered users.

Posted Thursday, June 26, 2003

Insight Graphing For C6

Insight Graphing is now available for Clarion 6.

Posted Thursday, June 26, 2003

RDraw Class To Be Released This Week

The first version of the new RDraw class will be released during this week. This drawing class is a Windows control with available properties and methods. The product will be released in stages and the first stage will only accept programatically setting the drawing control. The second and later releases will interact with the end user. The source code for the original Clarion classes is available for purchase for unrestricted commercial re-development and use.

Posted Wednesday, June 25, 2003

1stLogoDesign Fourth Of July Sale

Save 50% on the following products through July 4, 2003: Ready-to-Go Logos; Ready-to-Go Splash Screens; Ready-to-Go Specialty Images; Ready-to-Go IconsXP. Bundles are discounted 25%. Includes 60 day money back guarantee and a year of free upgrades. To get your discount, make your purchase online. Your card will not be charged automatically and the price in the total will be the regular price. After placing your order email jfmoreno@gitanosoftware.com requesting your discount, and it will be applied before your card is charged.

Posted Wednesday, June 25, 2003

gReg Fourth Of July Sale

gReg and add-ons are 50% off through July 4, 2003, including the competitive version and upgrades. Includes 60 day money back guarantee and a year of free upgrades. To get your discount, make your purchase online. Your card will not be charged automatically and the price in the total will be the regular price. After placing your order email jfmoreno@gitanosoftware.com requesting your discount, and it will be applied before your card is charged.

Posted Wednesday, June 25, 2003

Gitano Fourth Of July Sale

All single Clarion tools at the Gitano site are 50% off through July 4, 2003. Bundles are discounted 25% Includes 60 day money back guarantee and a year of free upgrades. To get your discount, make your purchase online. Your card will not be charged automatically and the price in the total will be the regular price. After placing your order email jfmoreno@gitanosoftware.com requesting your discount, and it will be applied before your card is charged.

Posted Wednesday, June 25, 2003

RPM For C6 EA4.5

A new install of RPM for C6EA4.5 is now available.

Posted Monday, June 23, 2003

ClarionTools Clarion 6 Support

Nice Touch Solutions plans to support Clarion 6 as soon as possible after the final release of Clarion 6. Those interested in the pre-release please send an email indicating what products you are interested in testing.

Posted Monday, June 23, 2003

SCA Microtemplates VAT On EU Sales Starting July 1st, 2003

SCA's online merchant, eSellerate, has just announced it will be charging VAT tax in sales to customers located in the EU. The actual percent will be 15-25% (depending on the member state that the end user resides in. This only applies to sales made using eSellerate, it does not apply if you pay using PayPal.

Posted Monday, June 23, 2003

LinderSoft EU-Based Customers And VAT

Due to a new European Union Directive, the Lindersoft WebStore must begin collecting value added tax (VAT) from customers in the European Union on 1 July 2003. Under the new rules, sales provided from countries outside of the European Union to non-taxable persons in the EU are no longer considered tax free because the place of taxation is shifted from the place the service provider or operator is established to the place where the recipient of the services is located.

VAT applies to customers who reside, have a permanent address, or are established in the European Union (EU). If you are located in the EU and not registered for Value Added Tax, buy now and save up to 25%.

Posted Monday, June 23, 2003

SetupBuilder 5.0 Beta Due Soon

Linder Software expects the first beta version of SetupBuilder 5 to ship within the next three weeks. The beta 1 version will go out to 700 qualified testers worldwide. The primary focus for the beta 1 phase will be the actual functionality the SetupBuilder 5 IDE (written in Clarion 6) provides. The second stage will provide the 32-bit compiler to generate native 32-bit script based installations. Remaining features, including the MultiVersion file updater, Application self-repair, Web install and Web update capability, as well as user documentation, are scheduled for the third phase.

Posted Friday, June 20, 2003

SimTabTree Template 1.04

Version 1.04 of the SimTabTree template is now available. The setting of the IMM attribute has been discontinued, and code has been added to allow the up arrow key to scroll smoothly past sheet controls. Registered users can download the updated version for free from the same URL they have used before. The opening special price of $19 US (normal price $29 US) will lapse on 30 June 2003. The template can be purchased from www.clarionshop.com. A Demo is also available from SimSoft.

Posted Wednesday, June 18, 2003

ClassViewer Web Pages Updated

The ClassViewer web pages have been updated, and now features screen shots showing the latest additions to the product, including a three-pane view that makes it easy to browse method source code.

Posted Wednesday, June 18, 2003

SimTabTree Template Updated

Eric Churton has made a small, but important change to the SimTabTree template, fixing a bug where the queues created had not been freed on closing the relevant procedure. The new version is now available for download. The template is available from www.clarionshop.com at the opening special price of $19 US (normal price $29 US). Demo available.

Posted Friday, June 13, 2003

Look Good Package Redesigned

Gitano's Look Good Package has been redesigned and now includes more goodies. This is not an upgrade, and if you would like to get the new additions you can do so by paying the difference. The new package now includes: Theme Pack 1; Icons XP 1 & 2 (over 320 icons in ICO and GIF format); gBuddy (create your own themes); 20% off on custom work. The new price is $99, and if you own any of the following a special discount applies: Theme Pack 1 -$30, your price $69; Icons XP (any) -$20, your price $79; gBuddy -$30, your price $69; Look Good Package -$80, your price $19.

Posted Thursday, June 12, 2003

Fenix: Beta 2.0 Shipped

Fenix Beta 2.0 (last beta before Gold) has been shipped as planned. New features include role based security, statement management, crystal reports integration, and sheet/tab support. Fenix integrates with .NET framework guidelines and recommendations to implement Role-Based security, which basically enables you to define access to a procedure/page only to authenticated users who have an existing role, but also to define more atomic restrictions like hiding or disabling a button (any control) for specific roles. Statement management lets you produce applications that will remember the most important information about your filters, locators, sort orders, etc. Fenix now will keep this information so that when you navigate through the system your navigation is made easier. Sheets/tabs are now mapped transparently to the developer.

Posted Thursday, June 12, 2003

Fomin Report Builder v.2.86

A new, C6 compatible release of Fomin Report Builder is now available. It is a free update as before. Registered users can get appropriate download information from the Live Update service at the web site. This product release is compatible with Clarion 4, 5, 5.5 and 6.

Posted Thursday, June 12, 2003

PD Browse Button Lookup Version 60-02 Released

PD Browse Button Lookup Version 60-02 (Beta) is now available for download. Changes and additions include a read only option, expansion of the disable on edit options, and a SqlFilter. This all source code class library is designed for C55 and Clarion 6.

Posted Thursday, June 12, 2003

Software Submission Service

Epsilon Concepts is now offering a Software Submission service. There are

several packages to choose from. Epsilon can generate the XML Pad File, submit your sites to many of the top download sites either that you specify or that are ranked highest in priority in Epsilon's system. Epsilon can also check the sites 30 days after submission to see if your software was listed and if not, submit again.

Posted Thursday, June 12, 2003

ClassViewer 6.0 Released

Keystone Computer Resources has released ClassViewer 6.0 - a utility program that combines the best features of the Clarion Application Builder Class Viewer and the Call Tree utility. In addition, ClassViewer 6.0 provides information on Interfaces, Structures and Equates. With ClassViewer 6.0 you can analyze all classes (not just ABC), make annotations, view the class source and help files, and use your favourite editor to maintain your classes. Features include: Full support for Clarion versions 2.0 through 6.0; View Classes, Interfaces, Call Tree, Structures, Equates; Split panel view with object list on the left and tree view on the right; Class filter options are All/ABC Only/Non-ABC Only; Filter classes by Category; Tree view shows hierarchy of Classes, Properties, Methods, and Interfaces; Ability to specify detail level for objects to provide additional filtering of the tree view; Option to include/exclude items with the protected and private attributes; Hyperlink capability to jump to referenced items; Easy hyperlink navigation; Add notes to all item types; View declaration and/or definition source files; Access the Clarion Help for classes and methods; From the CallTree view you can see a list of methods that call the selected method; View method calls in context from the CallTree view; Automatically scans all class inc and clw source files; Optionally scan additional files; Smart Scan technology that will only scan for changes once the initial scan is done; Application font and colors are user definable; Easily use your favorite editor to manage the source files; XML data output of raw data parsed by the application; XML output of the tree view. Complete help file included; free support and upgrades. Price is $99.00 USD.

Posted Thursday, June 12, 2003

SetupBuilder 4.03 Final Release Candidate

The final release candidate for SetupBuilder 4.03 Standard Edition is available now. The update is free of charge to all SetupBuilder Standard Edition customers. Version 4.03 will be the last 4.x release.

Posted Tuesday, June 10, 2003

Newsletter Service Update

You can now purchase the articles used in the Castle Computer Technologies directly. You can then edit, format and display the articles any way you like.

Each month you will receive an email with all the articles from that month's newsletter in editable plain text format. You can still purchase the newsletter pre-formatted in PDF format or have it printed for you. "Computers & You" is designed to be a marketing newsletter for your company. It is not technical and it is written for all levels of computer users.

Posted Saturday, June 07, 2003

RPM Email Survey

Lee White is looking for feedback, from current and prospective RPM users, about email support in RPM.

Posted Saturday, June 07, 2003

True Edit-In-Place Template

This new EIP Template adds full template support for the Clarion edit-in-place list box. ABC templates only; includes source and future updates.

Posted Saturday, June 07, 2003

SimTabTree Template Update

A new version of the SimTabTree template is now available. The template now makes it easier to select Sheet controls (other controls are excluded from the list to pick from) and also allows the programmer to set the Wizard and No-sheet properties through the template (which makes it simpler to work with your tabs without having to set these properties on the sheet itself.) Updates are free to existing users. The templates are available on special for $19 US (as opposed to $29 US) at ClarionShop until 30 June 2003.

Posted Saturday, June 07, 2003

Early Bird SQL Seminar Special Ends Soon

You have until Tuesday, June 10 2003 to get in on the early, early bird special for the Boston Clarion/SQL Server seminar. England is all but sold out and Boston is 80% sold.

Posted Saturday, June 07, 2003

RDRAW Demo

The demo of the RDRAW alpha version have been updated and shows end user flowcharting, a programmed flowchart and starting procedures off the flowchart objects. Instead of being released as pure Clarion Classes calling GDI methods, the product is being developed as a C style new windows class that can be accessed through the calling of its methods and properties. For the Clarion environment, a template and classes will be supplied with the product. In the future this product will move from the Clarion third party section to the

Commercial section.

Posted Saturday, June 07, 2003

Accounting System Demo

A 60 day demo for the all-Clarion UniQue Accounting System is now available for download.

Posted Saturday, June 07, 2003

New Permanent cpTracker Price

cpTracker is now permanently priced at $79.00. cpTracker Lite, the project/task management system subset, is $27.00

Posted Saturday, June 07, 2003

PlugIT Clarion 6 Compatible

PlugIT version 1.02 (the current shipping version) is now compatible with all versions of the Clarion 6 EA program.

Posted Saturday, June 07, 2003

CHT C55/C6 Co-compatible Build Coming July 31

A new, double build will be available to CHT subscribers for July 31, 2003 numbered O8A1.0. It will contain a C55/C6 co-compatible install. That is, the same code base will work with both C55H and C6. The template (and the underlying CHT classes) will be C55/C6 aware and will modify their behavior accordingly.

Posted Monday, June 02, 2003

File Manager 3 Beta 18 Released

File Manager 3 beta 18 is now ready and available for download. There have been quite a few new features added, as well as fixes and improvements of existing code. In some places, the default behaviour has changed from the previous release.

Posted Monday, June 02, 2003

EasyResizeAndSplit Special Offer

The first twenty (20) customers who buy EasyResizeAndSplit will receive a free copy of Product Scope 32 PRO V4.5(a), Spreadsheet Version, a multi-purpose tool used to display, create, organize, and research profile exchanges. Some currently available profile exchanges are Clarion 3rd Party, Search Engine, Home Theater, Newsgroup and Email, and PRO Music USB. Profile Exchanges are collections of data with common themes, categories or interest; designed to help you find information on the Internet more quickly in an organized fashion,

locating related data and products.

Posted Monday, June 02, 2003

CPCS C6 Beta for EA4-3 Released

New builds of CPCS for C6 Beta EA4-3 are now available.

Posted Monday, June 02, 2003

New SimTabTree Template Demo

The SimTabTree template demo has been amended to include two Before/After procedures. Each procedure shows a sheet and tabs as they might be before adding the template, and on the same window, the effect of the template on a duplicate sheet and tabs. The SimTabTree template is currently on special at $19 US (instead of the normal price of $29 US) and is available from www.clarionshop.com

Posted Monday, June 02, 2003

EasyAutoEntry Review At ClarionShop

There is new review of EasyAutoEntry available on the ClarionShop reviews page. EasyAutoEntry provides IntelliSense-type technology for end users. Requires Clarion 5.0b or Clarion 5.5, ABC or Legacy, 32 bit only.

Posted Monday, June 02, 2003

Beta Testers Wanted For A Clarion XML Document Reader

Steve Ryan is looking for beta testers for an XML document reader written in Clarion. The XML reader now supports basic DOM structure and SAX, including: Parsing a complete document; Returning elements of a document; Returning records set of a document; Element attributes; Parses command supports statements and splits parameters of comment sets. Alpha features under development include: Parsing DTD tables; Embedded support for runtime expression parsing and commands for processing and consolidating XML document based on a schema; Support for business enterprise services, SOAP services and business methods. This product will be available free to Clarion developers.

Posted Monday, June 02, 2003

Clarion Magazine

Creating A Designer Interface In Clarion (Part 3)

by David Podger and Deon Canyon

Published 2003-06-11

In the first and second parts to this article we introduced a touch screen application development environment for use in public health education projects. This application allowed an end user-designer (Deon, at first) to create a scripted touch screen application using a designer interface, written in Clarion, which is a variation on Clarion's own window formatter interface. Last time we looked at the code involved when the user selects a control with the mouse. In this third of three parts we'll show what happens when the user moves the mouse (with the left button held down) and then releases the button.

The MouseMove code

The `MouseDown` code sets the scene for `MouseMove` and `MouseUp`. The full `MouseMove` code follows:

```
! get mouse location after a move
NX = MOUSEX()
NY = MOUSEY()
! Groups in control-box cannot be moved or re-sized
IF CQ:Area <> 2
  ! saved X,Y position set, so movement can be measured
  IF SaveX
    ! not panels
    IF Border = '' AND CQ:WhichMove[1 : 2]<> 'Pa'
      XBox = (CQ:CX+(NX-SaveX))
      YBox = (CQ:CY+(NY-SaveY))
      WBox =  CQ:CW
      HBox =  CQ:CH
      IF ControlChain
        DO PositionBox      ! chained boxes to move
      .
    ELSIF Border = 'Bottom'
      XBox = CQ:CX
      YBox = CQ:CY
```

```
      WBox = CQ:CW
      HBox = (CQ:CH+(NY-SaveY))
    ELSIF Border = 'Right'
      XBox = CQ:CX
      YBox = CQ:CY
      WBox = CQ:CW+(NX-SaveX)
      HBox = CQ:CH
    ELSIF Border = 'Top'
      XBox = CQ:CX
      YBox = (CQ:CY+(NY-SaveY))
      WBox = CQ:CW
      HBox = (CQ:CH-(NY-SaveY))
    ELSIF Border = 'Left'
      XBox = (CQ:CX+(NX-SaveX))
      YBox = CQ:CY
      WBox = (CQ:CW-(NX-SaveX))
      HBox = CQ:CH
    ELSIF Border = 'BottomRight'
      XBox = CQ:CX
      YBox = CQ:CY
      WBox = (CQ:CW+(NX-SaveX))
      HBox = (CQ:CH+(NY-SaveY))
    ELSIF Border = 'TopRight'
      XBox = CQ:CX
      YBox = (CQ:CY+(NY-SaveY))
      WBox = (CQ:CW+(NX-SaveX))
      HBox = (CQ:CH-(NY-SaveY))
    ELSIF Border = 'BottomLeft'
      XBox = (CQ:CX+(NX-SaveX))
      YBox = CQ:CY
      WBox = (CQ:CW-(NX-SaveX))
      HBox = (CQ:CH+(NY-SaveY))
    ELSIF Border = 'TopLeft'
      XBox = (CQ:CX+(NX-SaveX))
      YBox = (CQ:CY+(NY-SaveY))
      WBox = (CQ:CW-(NX-SaveX))
      HBox = (CQ:CH-(NY-SaveY))
    END
```

In the above code, the movement of a whole control and the eight different ways of dragging one of its borders are dealt with. The last is the most complex since, when the top left corner is dragged, everything changes. The variables `Xbox`, `Ybox`, `Wbox` and `Hbox` are changed to track the movement box's position and size changes.

In the code that follows, a control has been selected and has red handles. The designer is moving the mouse to select a particular border. The code is bypassed if any controls have blue handles. When a border is detected, the correctly shaped cursor is displayed, indicating that it can be selected and dragged and the `Border` variable is set. The code is written to find a handle so long as the mouse cursor gets close enough. The variables `H1` through `H8` are the red

handles.

```
ELSIF ~ControlChain    ! otherwise, which border is it?
  IF  NX >= ?H1{PROP:XPos}   AND |
      NX <= ?H1{PROP:XPos}+3 AND |
      NY >= ?H1{PROP:YPos}   AND |
      NY <= ?H1{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeNWSE)
    Border = 'TopLeft'
  ELSIF NX >= ?H2{PROP:XPos}    AND |
        NX <= ?H2{PROP:XPos}+3  AND |
        NY >= ?H2{PROP:YPos}    AND |
        NY <= ?H2{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeNESW)
    Border = 'TopRight'
  ELSIF NX >= ?H3{PROP:XPos}    AND |
        NX <= ?H3{PROP:XPos}+3  AND |
        NY >= ?H3{PROP:YPos}    AND |
        NY <= ?H3{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeNESW)
    Border = 'BottomLeft'
  ELSIF NX >= ?H4{PROP:XPos}    AND |
        NX <= ?H4{PROP:XPos}+3  AND |
        NY >= ?H4{PROP:YPos}    AND |
        NY <= ?H4{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeNWSE)
    Border = 'BottomRight'
  ELSIF NX >= ?H5{PROP:XPos}    AND |
        NX <= ?H5{PROP:XPos}+3  AND |
        NY >= ?H5{PROP:YPos}    AND |
        NY <= ?H5{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeNS)
    Border = 'Top'
  ELSIF NX >= ?H6{PROP:XPos}    AND |
        NX <= ?H6{PROP:XPos}+3  AND |
        NY >= ?H6{PROP:YPos}    AND |
        NY <= ?H6{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeNS)
    Border = 'Bottom'
  ELSIF NX >= ?H7{PROP:XPos}    AND |
        NX <= ?H7{PROP:XPos}+3  AND |
        NY >= ?H7{PROP:YPos}    AND |
        NY <= ?H7{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeWE)
    Border = 'Left'
  ELSIF NX >= ?H8{PROP:XPos}    AND |
        NX <= ?H8{PROP:XPos}+3  AND |
        NY >= ?H8{PROP:YPos}    AND |
        NY <= ?H8{PROP:YPos}+3
    SETCURSOR(CURSOR:SizeWE)
    Border = 'Right'
  ELSE
    ! display normal cursor if not near a border
    SETCURSOR()
        ! set as no border
```

```
    Border = ''
  . . .
```

The MouseUp Code

MouseUp code is particular to each control and cannot be shown in detail. Some controls, however, get very similar treatment and can be used to present the general methodology. The code described above (`MouseMove`) shows only the movement of one or more gray outline boxes, but in `MouseUp` code the movement is completed and the control itself is displayed at its new position and with its new size.

```
IF SaveX
  ! an X,Y position has been saved, so something can happen
  MX = MOUSEX()
  MY = MOUSEY()
  ! get selected Control record from ControlQ
  CQ:Control = SaveControl
  CQ:Index = SaveIndex
  GET(ControlQ,CQ:Control,CQ:Index)

! The code begins by excluding controls that need special treatment.
! For the remaining standard controls, the code completes whole-chain
! whole-control or border movement.

IF  CQ:WhichMove <> 'Check'     AND |
    CQ:WhichMove <> 'Button'      AND |
    CQ:WhichMove <> 'Prompt'      AND |
    CQ:WhichMove <> 'Navig'       AND |
    CQ:WhichMove <> 'Ctrl'        AND |
 CQ:WhichMove <> 'Group'       AND |
 CQ:WhichMove <> 'PBox'        AND |
    CQ:WhichMove[1 : 2] <> 'Pa'
  IF Border = ''
    ! whole control being moved
    CQ:CX += (MX - SaveX)
    CQ:CY += (MY - SaveY)
    IF ControlChain
      ! first deal with primary control
      DO SetRedHandles
      CQ:Area = CQ:CW * CQ:CH
      PUT(ControlQ,CQ:Control,CQ:Index)
      SETPOSITION(CQ:Control,CQ:CX,CQ:CY,CQ:CW,CQ:CH)
      ChainX = (MX - SaveX)
      ChainY = (MY - SaveY)
          ! lastly, reposition and re-size chain
      DO MoveChain
    .
  ELSIF Border = 'Bottom'
    CQ:CH += (MY - SaveY)
  ELSIF Border = 'Right'
    CQ:CW += (MX - SaveX)
  ELSIF Border = 'Top'
```

```
      CQ:CY += (MY - SaveY)
      CQ:CH -= (MY - SaveY)
    ELSIF Border = 'Left'
      CQ:CX += (MX - SaveX)
      CQ:CW -= (MX - SaveX)
    ELSIF Border = 'BottomRight'
      CQ:CW += (MX - SaveX)
      CQ:CH += (MY - SaveY)
    ELSIF Border = 'TopRight'
      CQ:CY += (MY - SaveY)
      CQ:CW += (MX - SaveX)
      CQ:CH -= (MY - SaveY)
    ELSIF Border = 'BottomLeft'
      CQ:CX += (MX - SaveX)
      CQ:CW -= (MX - SaveX)
      CQ:CH += (MY - SaveY)
    ELSIF Border = 'TopLeft'
      CQ:CX += (MX - SaveX)
      CQ:CY += (MY - SaveY)
      CQ:CW -= (MX - SaveX)
      CQ:CH -= (MY - SaveY)
    .
    IF Border <> '' OR ~ControlChain
      ! move whole control
      DO SetRedHandles
      CQ:Area = CQ:CW * CQ:CH
      PUT(ControlQ,CQ:Control,CQ:Index)
      SETPOSITION(CQ:Control,CQ:CX,CQ:CY,CQ:CW,CQ:CH)
. .
CASE CQ:WhichMove
  OF 'Question'
    DQUE:QuestionX = CQ:CX
    DQUE:QuestionY = CQ:CY
    DQUE:QuestionW = CQ:CW
    DQUE:QuestionH = CQ:CH
    DO ShowButtonPosition
  OF 'QBox'
    DQUE:QBoxX = CQ:CX
    DQUE:QBoxY = CQ:CY
    DQUE:QBoxW = CQ:CW
    DQUE:QBoxH = CQ:CH
    DO ShowButtonPosition
  .
  ! Controls needing special treatment are dealt with
  ! here (code not shown).

. . .
```

The updating of `DQUE:` variables, saves their position and size. The routine `ShowButtonPosition` updates the control-box with `X`,`Y`,`W` and `H` values for the control. For example:

```
ShowButtonPosition
IF MoveQuestion
  ButnX = DQUE:QuestionX
```

```
  ButnY = DQUE:QuestionY
  ButnW = DQUE:QuestionW
  ButnH = DQUE:QuestionH
  DISPLAY(?ButnX,?ButnH)
ELSIF MoveQBox
  ButnX = DQUE:QBoxX
  ButnY = DQUE:QBoxY
  ButnW = DQUE:QBoxW
  ButnH = DQUE:QBoxH
  DISPLAY(?ButnX,?ButnH)
.
```

The variables `ButnX`, `ButnY`, `ButnW` and `ButnH` display positions in the control-box for the currently selected control. The routine `SetRedHandles` displays red handles.

```
SetRedHandles           ROUTINE
 SETPOSITION(?H1,CQ:CX+1,CQ:CY)                    ! top left
 SETPOSITION(?H2,CQ:CX+CQ:CW-4,CQ:CY)              ! top right
 SETPOSITION(?H3,CQ:CX+1,CQ:CY+CQ:CH-5)            ! bottom left
 SETPOSITION(?H4,CQ:CX+CQ:CW-4,CQ:CY+CQ:CH-5)      ! bottom right
 SETPOSITION(?H5,CQ:CX+(CQ:CW/2)-3,CQ:CY)          ! top middle
 SETPOSITION(?H6,CQ:CX+(CQ:CW/2)-3,CQ:CY+CQ:CH-5) ! bottom middle
 SETPOSITION(?H7,CQ:CX+1,CQ:CY+(CQ:CH/2)-3)        ! left middle
 SETPOSITION(?H8,CQ:CX+CQ:CW-4,CQ:CY+(CQ:CH/2)-3) ! right middle

! for special controls, some handles are disabled:

 ! initialise all handles
 HIDE(?H1,?H8)
 IF CQ:WhichMove = 'Check' OR |
  CQ:WhichMove = 'Ctrl'
   UNHIDE(?H6)                                      ! bottom middle
   UNHIDE(?H8)                                      ! right middle
 ELSIF CQ:WhichMove = 'PanelTop'
 UNHIDE(?H6)                                        ! bottom middle
 ELSIF CQ:WhichMove = 'PaBottom'
 UNHIDE(?H5)                                        ! top middle
 ELSIF CQ:WhichMove = 'PaneLeft'
 UNHIDE(?H8)                                        ! right middle
 ELSIF CQ:WhichMove = 'PanRight'
 UNHIDE(?H7)                                        ! left middle
 ELSE
 UNHIDE(?H1,?H8)                                    ! unhide all
 .
```

The control-box

A special control that deserves some further explanation is the moving, re-sizable control-box, shown in Figure 5:

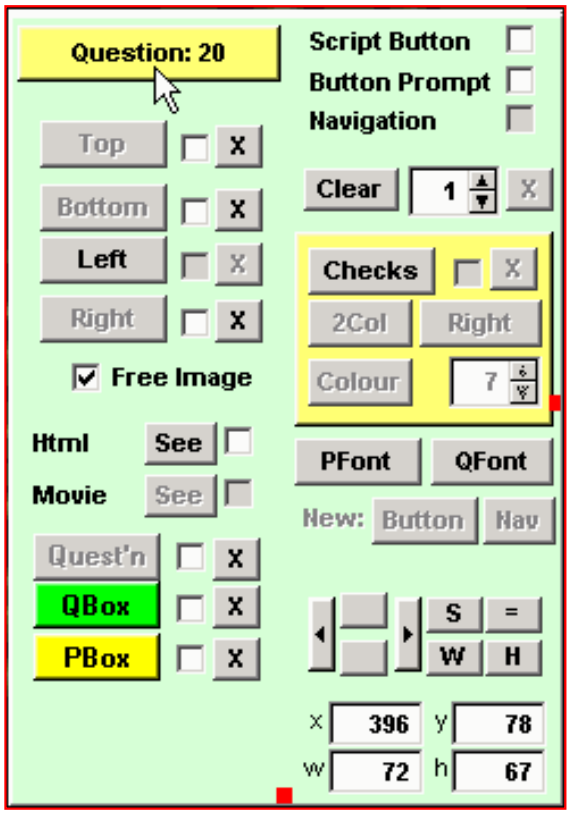

Figure 5. The moving, resizable toolbox

"Question: 20" above reminds the user which script question is being worked on. Clicking on **Question:20** (which does double duty as a button) changes the box's shape and re-arranges its controls. It is important to note that all the controls in the control-box are conventional Clarion controls, created within an entirely ordinary Clarion Window. The Clarion run time environment itself keeps track of where these controls are, even while the user is moving them about. As a result, the buttons in the control-box continue to work as normal buttons no matter where they happen to be on the screen.

When the user clicks within the control-box, but *not* on any particular control, red handles appear around it and it may be dragged about and re-sized. Here is another view of the tool box, after one re-sizing step. Red handles to allow re-sizing are clearly visible:

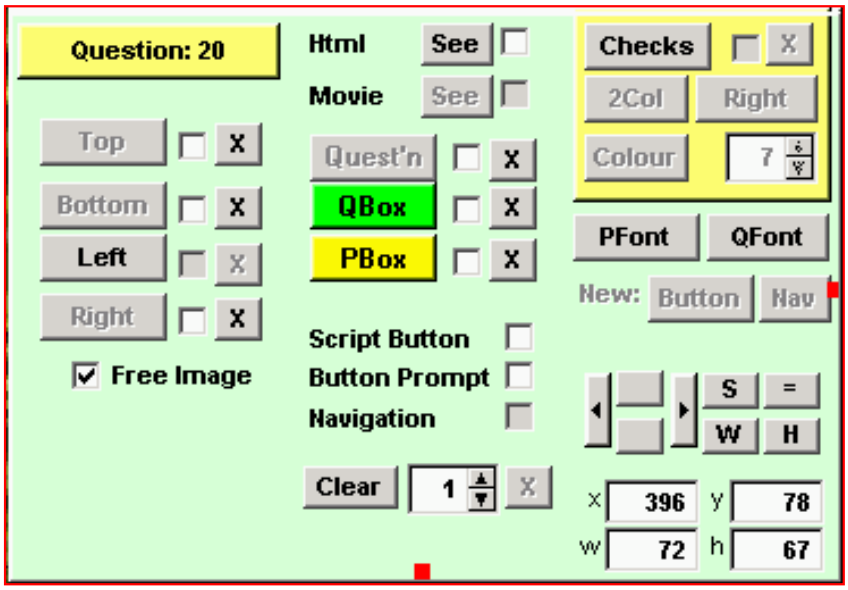

Figure 6. Another view of the toolbox

The control-box (like a CLARION toolbox) may be dragged so that most of it is off-screen, right out of the way of the designer.

In Summary

A graphic design interface can be written in a few pages of Clarion code. Such an interface opens the door to allowing users to design parts of their applications, such as the layout of reports and windows. In our case, the interface is developed further, to the point where it becomes (in combination with a scripting engine) a simple language in which to design educational touch screen applications.

David Podger has been an independent Clarion developer in Australia for a decade. He presently lives in Katherine in the Northern Territory, where he sells a specialised accounting application for remote communities.

Dr Deon Canyon is a Lecturer (translation: Assistant Professor) in public health and medical entomology at James Cook University in North Queensland, Australia. His current research interests include: Disease control simulation models for use in distance education; Lymphatic filariasis (a disease caused by thread-like parasitic worms) and the Aedes polynesiensis mosquito in transmission and control of the Pacific Head Lice (Pediculus capitis); Health in remote settlements and the use of touch screen kiosks to improve it; Aedes aegypti mosquito behavior, physiology and ecology . Deon is 38, married, and has three children. He likes rock climbing but busted his knee and can't do it anymore.

Reader Comments

Add a comment

Clarion Magazine

Many Reports: Many Printers

by Henry Plotkin

Published 2003-06-13

Frankly, I thought the request was just plain stupid. My customer told me he used different printers for labels, end-of-day reports and other reports. But he wanted these all to be "set up." "Pre-configured," I guess describes it accurately.

One of the things Windows does is allow us to change printers whenever we want. Wizarded Clarion application frames (main procedures) come with a **Print Setup** menu item already populated, which calls the standard Windows printer selection dialog. I am also a long time RPM user and RPM allows changing printers just before a report is run, as shown in Figure 1.

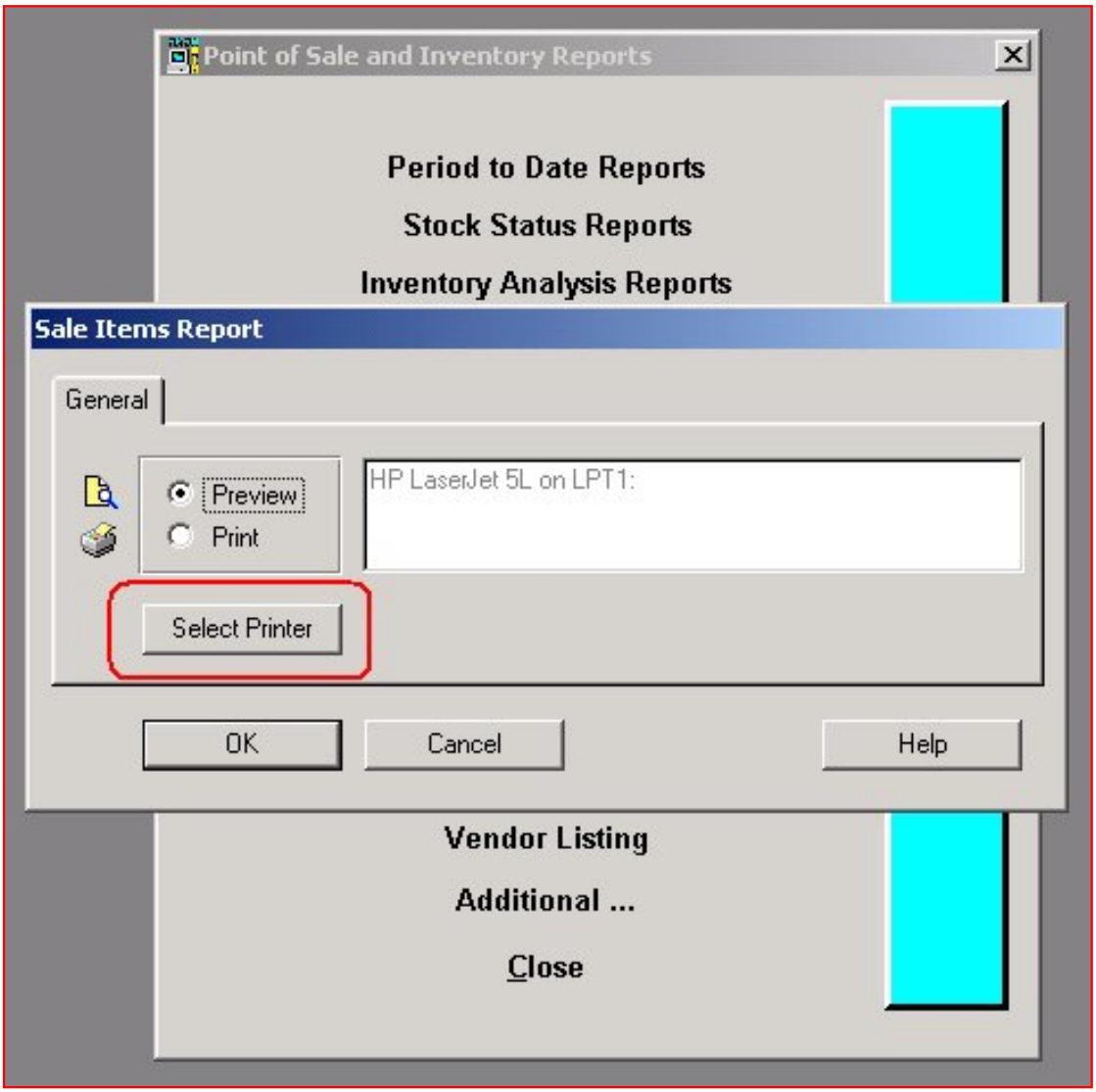

Figure 1. RPM pre-print dialog

In other words, my customer has three opportunities to change printers before printing a report: in Windows itself, from the Print Setup menu, and from the RPM preview.

Of course, I also wondered how he had three printers available to him on a regular basis. Two, it so happens, are network printers. Well, I'd never realized I had computer-literate customers.

So, this guy is entitled to a little respect. Maybe his request isn't as stupid as I thought. Besides, making my software a little easier to use would also make it more marketable.

I decided to add an additional menu item, something like the EOD & Label Printers item in Figure 2.

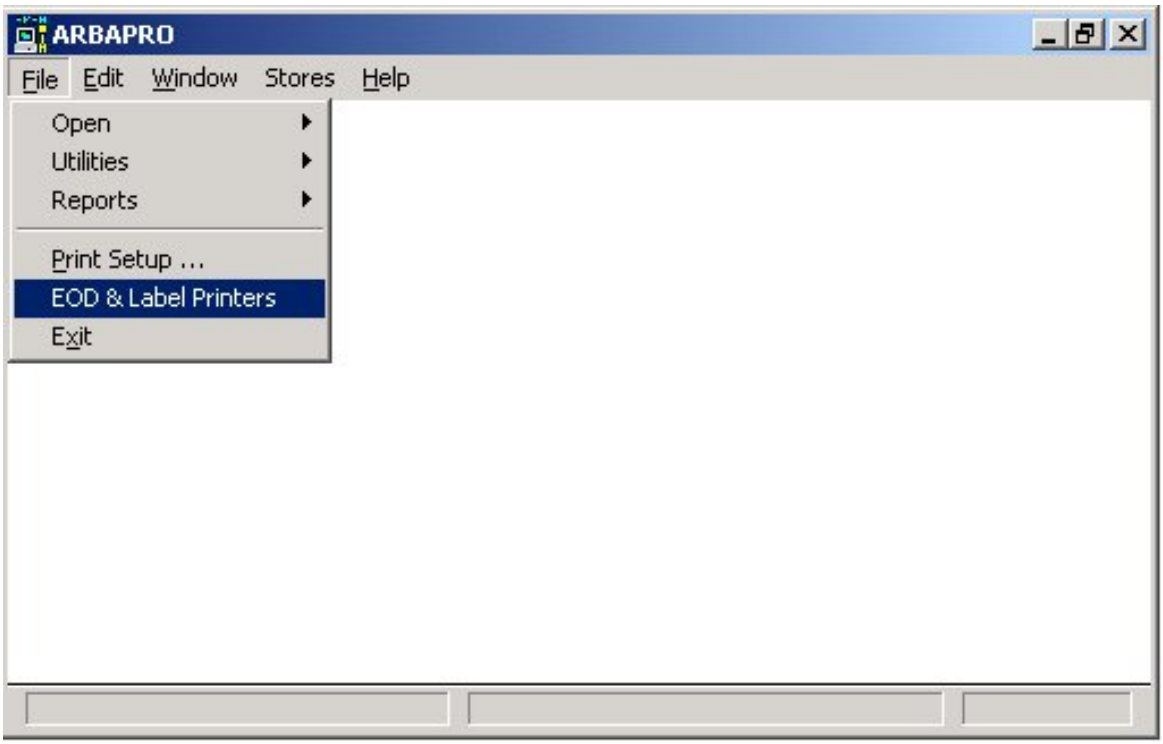

Figure 2. New menu item for function-specific printers

Design Parameters

Clarion provides a `PrinterDialog` statement. The Clarion Help says that `PrinterDialog`

> displays the Windows standard printer choice dialog box (or the Print Setup dialog) and returns the printer chosen by the user in the PRINTER "built-in" variable in the internal library. This sets the default printer used for the next REPORT opened.

My plan was to use the default printer for reports other than labels and end-of-day reports and use `PrinterDialog` to set variables for the other report types. Figure 3 shows a simple printer selection window.

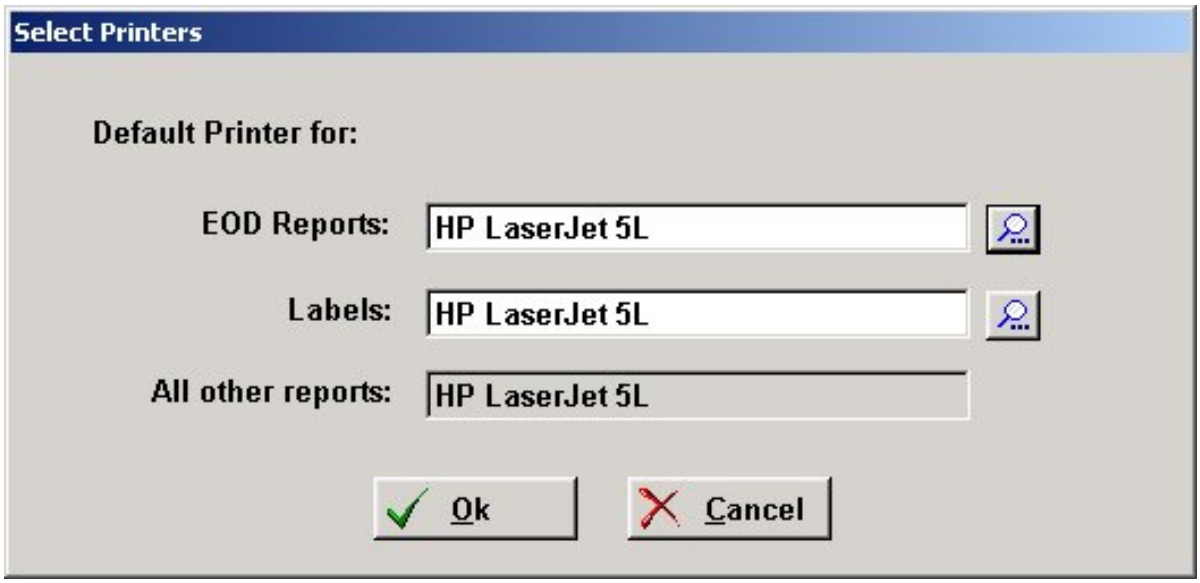

Figure 3. Selecting printers for various jobs

This window shows the contents of two global variables, `GLO:EODPrinter` and `GLO:LabelPrinter`, which are read-only. This prevents the user from making any text entry.

I do not want to allow data entry because I want to be absolutely certain that the values in the variable actually represent printers installed and configured on the user's computer. Instead, the lookup button to the right of each entry field calls `PrinterDialog` to load the appropriate variable. Here's the code for the first button:

```
PrinterDialog('Choose Printer for EOD Reports')
GLO:EODPrinter = Printer{PropPrint:Device}
Display
Printer{PropPrint:Device} = Printer
```

The other button uses similar code (the prompt and variable are different, that's all).

The last line of this code bears some explanation. The Printer variable contains the windows default printer (I capture this data when the window opens). The last line restores the windows default printer because if I didn't, the next report to print would use the selected printer.

In fact, when the window opens, not only do I capture the default printer, I also store the current values of the two global printer variables:

```
Printer = Printer{PropPrint:Device}
SAV:EODPrinter = GLO:EODPrinter
SAV:LabelPrinter = GLO:LabelPrinter
```

If the user cancels this window, I am able to restore the previous values:

```
GLO:LabelPrinter = SAV:LabelPrinter
GLO:EODPrinter = SAV:EODPrinter
Post(Event:CloseWindow)
```

This code goes in the window's **Cancel** button embed (I'm using a simple window, so I have to take care of closing it myself).

Implementation

What I need to do now is to assign those printers when I run specific reports. Using Printer Control Properties (search for "PropPrint" in the on-line help), this is easy enough. First, I save the current printer. Then I set the appropriate printer:

```
SAVE:printer = PRINTER{PROPPRINT:Device}
Printer = GLO:EODPrinter
```

When the report is done, I reverse the process:

```
PRINTER{PROPPRINT:Device} = SAVE:printer
```

The real trick is to do this without calling PrinterDialog and that is just what this code does (see the on-line FAQ "How to change the printer device without calling PRINTERDIALOG"). Using the property syntax, I can set and re-set the `Printer` built-in variable. This is the same variable the user effects with PrinterDialog.

Summary

My customer's request first struck me as kind of dumb. But knowing what tools Clarion makes available (PropPrint and the FAQs) allowed me to make my customer happy with very little work.

"hp" in fact prefers Hewlett-Packard printers but will use whatever is available. Born in New York City, hp is a self-taught Clarion developer doing a sustantial amount of work for hospital gift shops.

Reader Comments

Add a comment

Heeheehee. Henry, we have our stuff setup so that they can...

Clarion Magazine

Creating An XML RSS Web Site Summary With Clarion 6 (Part 2)

by David Harms

Published 2003-06-13

In last week's installment I introduced the RSS XML file specification, and previewed some code I used to create an RSS file in Clarion 6. This week I'll go into that code in a bit more detail, and also discuss some of the problems and pitfalls of creating XML.

First, however, I'm pleased to announce that you can now get RSS feeds from Clarion Magazine! These are RSS 0.91 compliant and are available at the following URLs:

http://www.clarionmag.com/news.rss
http://www.clarionmag.com/articles.rss

The first link displays the 15 most recent news items posted on the Clarion Magazine site, and the second the 15 most recent articles. I'll be adding links on the home page soon.

REMINDER: You probably won't get a lot of joy out of viewing those links with your browser. These are XML files meant to be read by RSS readers. See this list of available RSS readers (my thanks to Mike Pickus) for more information.

One excellent use of RSS feeds is as a replacement for email notification. Clarion Magazine will continue to maintain mailing lists, but with these feeds you no longer have to wait for the weekly email summary. Just point your RSS reader at the ClarionMag RSS feeds you want and tell it to poll them on a regular basis (every hour should be more than sufficient).

Here's what one instance of news.rss, truncated from 15 to two items for the sake of brevity, looks like:

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="0.91">
  <channel>
    <title>Clarion News</title>
    <description>News, product announcements, and
      other items of interest to Clarion
      developers</description>
    <language>en-us</language>
    <link>http://www.clarionmag.com</link>
    <copyright>Copyright 1999-2003 by
      CoveComm Inc.</copyright>
    <item>
      <title>RPM Email Survey</title>
      <link>http://www.cwaddons.com/email/</link>
      <description>Lee White is looking for feedback,
        from current and prospective RPM users, about email
        support in RPM.</description>
    </item>
    <item>
      <title>True Edit-In-Place Template</title>
      <link>http://www.audkus.dk</link>
      <description>This new  EIP Template adds full
        template support for the Clarion edit-in-place
        list box. ABC templates only; includes source and
        future updates.</description>
    </item>
  </channel>
</rss>
```

Figure 1 shows the same RSS file (only not truncated) as displayed by NewzCrawler.

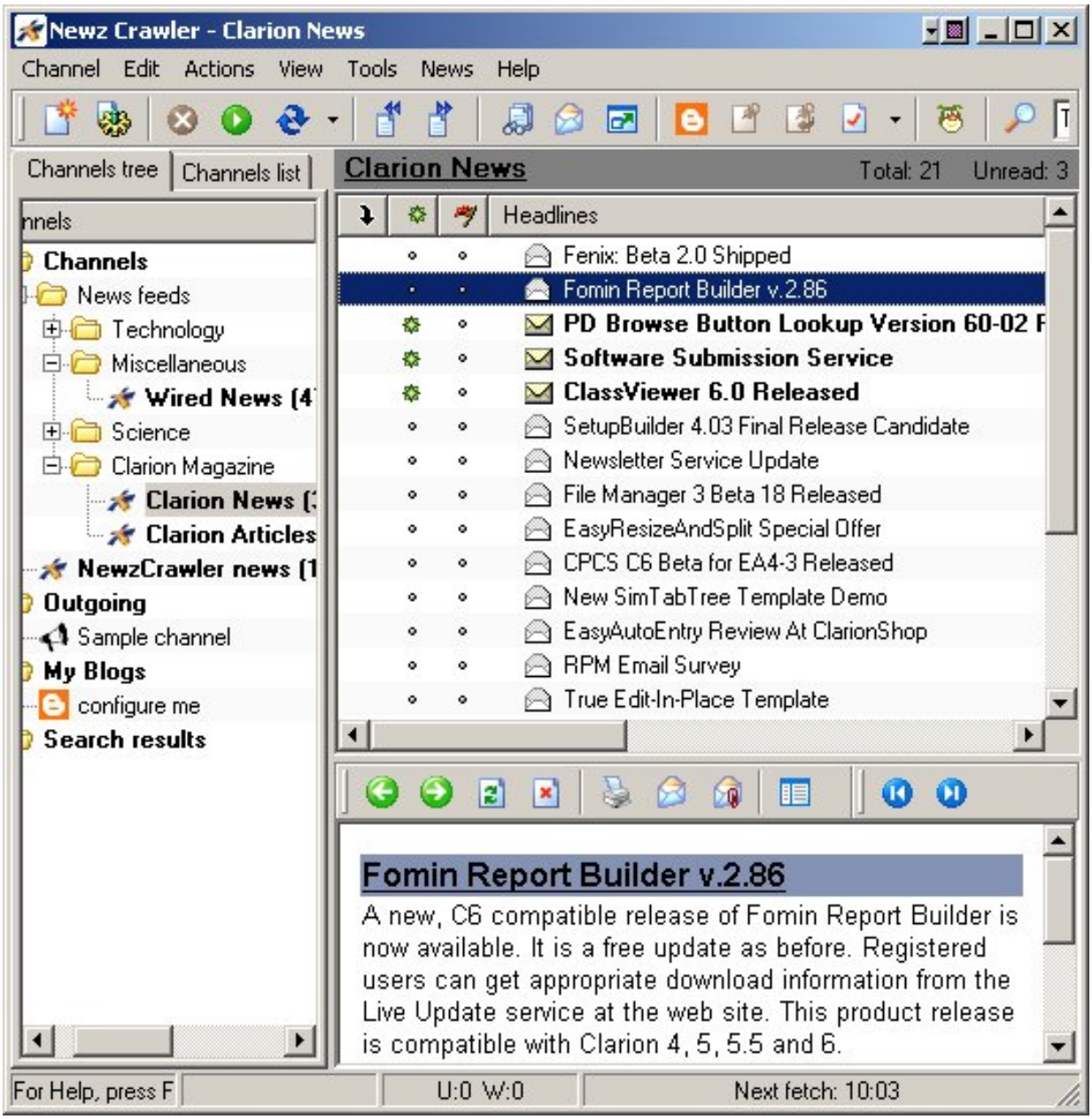

Figure 1. NewzCrawler displaying news.rss

I have a dirty little secret, however. Although this article is about how to generate XML in Clarion 6, the RSS files you get from the ClarionMag site are actually generated by server-side Java code, using JDOM, which is a class library that does a fairly good job of making DOM easy to use. After all, ClarionMag runs on a Linux box and Clarion code isn't an option there!

But never mind the Java code – this article really is about Clarion. And as I indicated last week, Clarion 6 really does have XML capabilities, even if they're not that well documented yet.

The sample app

You can download the sample application at the end of this article, or you can follow along by copying the source from this page. You will, however, need Konrad Byers' AnyASCII class to overcome one weakness in the C6 XMLGenerator class (at least as of EA4-4) – I'll cover that in more detail later. For now, unzip the AnyASCII files into the Clarion 6 libsrc directory.

Create a new C6 application, or use an existing app. Go to the Global Embeds list, and include the following in After Global Includes:

```
include('abprxml.inc'),once
include('anyascii.inc'),once
```

The first line includes the C6 `XMLGenerator` class, and the second includes Konrad's `AnyASCII` class.

Now create a new source procedure, or adapt the following code to an existing procedure. In the procedure's data section include the following:

```
rss    XMLGenerator
ascii  AnyAsciiFileClass
text   cstring(1000)
```

The rss variable is an instance of the `XMLGenerator` class – there is no `&` before the data type so `rss` is *not* a reference, but an automatically instantiated object that is ready to use when your procedure code runs; similarly `ascii` is also automatically instantiated. The `text` variable is used by the ascii object to store one line of text in the file. More about that in a bit.

There are three logical parts to the XML export code in the sample app. The first part initializes the `XMLGenerator` and creates the standard channel description tags:

```
rss.Init('rssTest.xml')
rss.OpenDocument('rssxxxxxxxxxxxxxxx')
rss.XMLVersion = '1.0'
rss.AddXMLHeaderAttribute('encoding','UTF-8')
rss.AddTag('channel','')
rss.AddTag('title','Clarion News','channel')
rss.AddTag('description','News, product announcements, and ' |
    & 'other items of interest to Clarion developers','channel')
rss.AddTag('language','en-us','channel')
rss.AddTag('link','http://www.clarionmag.com','channel')
rss.AddTag('copyright','Copyright 1999-2001, xmlhack team.','channel')
```

It's not too hard to see how the method calls work. First, you initalize the `XMLGenerator` object with the name of the file to be created. Then you create

the root tag.

The root tag

The root tag is an important concept in XML file processing. All XML files are made up of nested tags, and the topmost (or leftmost, if you prefer) tag is called the root tag. In fact, since any tag will know about its child tags, once you have the root tag you have a way to get at the entire document.

In the case of `XMLGenerator`, the parameter to `OpenDocument` is the root tag's name. All RSS files that follow the 0.91 specification have a root tag that looks like this:

```
<rss version="0.91">
```

The version is a tag attribute, and while you can create attributes for all other tags, in the current version of `XMLGenerator` you cannot yet create a root tag attribute. Accordingly, I'm going to have to process this file a second time using Konrad's `AnyASCII` class (you could also use the ASCII driver). Because I can't easily insert text in the middle of an ASCII file, I've chosen to make the root tag's name long enough to include the version attribute; I'll simply replace the text when it comes time to do so.

Referencing tags

Many XML implementations use objects to represent tags; that is, you create a root tag, and then you add other elements (tags) to that root tag, and so on down the hierarchy. Internally, XMLGenerator uses a tag queue, which can contain a reference to a queue of child tags, which can each contain a reference to a queue of child tags, which can…well, you get the idea.

Externally, however, you only use tag *names*, not tag *objects*, when creating the XML file. Here's the prototype for the AddTag method:

```
AddTag PROCEDURE(STRING pName,STRING pValue,<STRING pParent>)
```

The first parameter is the tag name, the second its value. The third and optional parameter is the name of the parent tag, if any.

The first tag under the root tag in the RSS file is a channel tag:

```
<channel>
```

Here's the code to create that tag:

```
rss.AddTag('channel','')
```

The second string, which would be the tag's value, is an empty string. If you do pass a value, say param2, it will appear like this:

```
<channel>
param2
```

But you want child tags here, not tag data. You create just the channel tag first, and because you don't specify the third parameter it becomes, by default, a child of the root tag. After creating the channel tag, create the child tags by specifying `'channel'` as the third parameter:

```
rss.AddTag('title','Clarion News','channel')
```

Creating the items

Once you've added all the channel tags you're ready to create the items described by the RSS feed. Here I'm just going through the motions; in a real-world situation, you'll be looping through a file/table, so you'll probably want to integrate this code into a process:

```
rss.AddTag('item','','channel')
rss.AddTag('title','RPM Email Survey','item')
rss.AddTag('link','http://www.cwaddons.com/email/','item')
rss.AddTag('description','Lee White is looking for feedback, ' |
  & 'from current and prospective RPM users, about email ' |
  & 'support in RPM.','item')
```

The first AddTag call adds an empty `'item'` tag to the channel tag. The subsequent `AddTag` calls add the child tags, along with their data.

Finally, close the document:

```
rss.CloseDocument()
```

Now comes the fun part. Because there's no way (as of this writing) to set a root tag attribute, I now process the file a second time using Konrad Byers' AnyASCII class:

```
! Once XMLGenerator supports root tag attributes
! the following hack will not be needed
if text = '<rssxxxxxxxxxxxxxxx>'
   ascii.Replace('<rss version="0.91">')
```

```
   loop
      if ascii.Read(text) <> level:benign then break.
      !message(text)
      if text =  '</rssxxxxxxxxxxxxxxx>'
         ascii.Replace('</rss><!--ignoreme-->')
      end
   end
end
```

This code replaces the placeholder text in the root tag with the version information. Of course, the end tag will also have the same name. You could either replace the first `x` with `>` and truncate the file, or do as I have done and replace the unneeded characters with an XML comment.

Note that Konrad's class is not yet C6 compatible, so I took the liberty of removing the code that automatically links in the ASCII driver.

Why use XMLGenerator?

`XMLGenerator` isn't the most full-featured XML export class in the world, but it is easy to use. And I trust that as time goes on it will get the features it's still missing, such as root tag attributes and pretty printing (proper indents).

There are other reasons to use XMLGenerator over a simple ASCII driver. Keep in mind that although you can create your own tags in XML, tag names and tag values must adhere to certain standards. Not all characters are permitted. Tag names must conform to certain rules. The `XMLGenerator` class has a method called `MakeValidName`, which the classes uses to ensure valid tag names. This method will replace any of the following characters, plus the space character, with an underscore.

```
. [ ] {') } + = '') " ! @ # $ % ^ & * ( ) ; , < > ? / | \ ~ `')
```

If namespaces are not in use (a subject beyond the scope of this article) then `MakeValidName` will also replace the : character with an underscore.

You also have to be careful with tag values. For instance, I've had problems with articles.rss where I copied some text with a typical, curved apostrophe character (') into a word document. That resulted in a badly formed RSS document because the character code of ' was outside the allowed range for tag values. If you can't get rid of your invalid characters, your next option (think of it as more of a last resort) is to use a CDATA section. Unfortunately XMLGenerator doesn't yet support CDATA sections. You would typically use a CDATA section when including, say, some HTML or XML inside an XML document.

Your alternative, if you really need CDATA sections, is to dive into the Clarion 6 wrapper for the CenterPoint parser (see abprmxl.clw and abprxml.inc). That is also beyond the scope of this article, but I hope to find some time to go exploring there soon.

I also found that my use of non-breaking spaces in article titles caused problems. The title of this article, for instance, is stored in the database as "Creating An XML RSS Web Site Summary With Clarion 6 (Part 2)". I do that so that if just the end of the title wraps I won't get an orphaned "2)" on the next line. I had to write some code to strip out the non-breaking space and replace it with a normal space character.

Validating RSS

So how do you know if your RSS document is valid? Just go to one of these online validators (I'm sure there are many others) and submit the URL for your RSS document:

Userland
http://aggregator.userland.com/validator

Experimental Online RSS 1.0 Validator
http://www.ldodds.com/rss_validator/1.0/validator.html

The internet data feed archive
http://feeds.archive.org/validator/

Make sure the validator you're using supports your chosen version of RSS, and also test against a few validators. I've noticed some inconsistencies during testing, where a document was given the stamp of approval by one validator, and rejected by another. You may also see some variations in how well the various RSS readers handle bad data.

Summary

RSS is a relatively simple application of XML, but it's generating a lot of excitement because it provides a standard way of presenting web site summary information. There are numerous RSS readers (a.k.a. aggregators) available which can help you manage the growing number of RSS feeds, including those provided by Clarion Magazine.

Although Clarion 6's current (EA4-4) release of XMLGenerator is still missing a few useful features, it promises to be an easy way to create a wide variety of XML documents, many no doubt much more complex than RSS.

Download the source

Download the AnyASCII class (modified for C6)

XML/RSS Resources

XML's home page on the web
http://www.w3.org/XML/

The XML FAQ
http://www.ucc.ie/xml/

XML.org
http://www.xml.org/

Useful RSS links from the government of Utah
http://gils.utah.gov/rss/

The RSS FAQ
http://www.voidstar.com/module.php?id=129&mod=book&op=feed

A list of RSS readers/clients
http://www.ourpla.net/cgi-bin/pikie.cgi?RssReaders

Microsoft-related RSS feeds, including MS developers
http://www.microsoft-watch.com/article2/0,4248,1097192,00.asp

The Userland RSS 0.91 page
http://backend.userland.com/rss091

The Userland RSS Feeds list
http://backend.userland.com/directory/167/feeds

Radio Community Server Top 100 RSS Feeds
http://radio.xmlstoragesystem.com/rcsPublic/rssHotlist

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of Developing Clarion for Windows Applications, published by SAMS (1995). His most recent book is JSP, Servlets, and MySQL, published by HungryMinds Inc. (2001).

Reader Comments

Add a comment

I forgot to mention that Konrad's AnyASCII class is not...

Clarion Magazine

Topics

Clarion Magazine's RSS Feeds

You may have noticed a couple of new icons on the ClarionMag home page. These XML buttons are links to Clarion Magazine's RSS feeds. What's an RSS feed, you say? Well, if you've been (ahem!) reading my articles on creating RSS feeds in Clarion, you'd know that an RSS feed is an XML file that provides web site summary information. RSS is variously interpreted to stand for RDF Site Summary, Rich Site Summary, or Really Simple Syndication.

It really doesn't matter what you call it; what matters is that RSS is rapidly becoming a vital tool for managing information on the Internet, and on Intranets. With RSS you don't need to visit a web site to see what's new, and you don't need to register yourself with a web site and wait for them to send you an update email. Instead, you use a program called an RSS reader or aggregator (there are many available); that program reads the RSS feeds you specify, and presents you with a summary of that information. Of course those web sites must have RSS feeds, and Clarion Magazine is now among that number!

Some of the uses for RSS feeds include:

- News items
- Web site changes
- Product announcements
- Weblog summaries

You can probably think of other uses. At present Clarion Magazine has two RSS feeds:

- 15 most recent articles XML
- 15 most recent news items XML

You can click on either link to see the XML, but really you want an RSS reader or aggregator to make the best use of the feed. Point the program of your choice

at these two feeds, set them to be read every hour or so, and you'll be on top of the latest happenings at Clarion Magazine!

If you already have an RSS reader, and you're looking for some other feeds to follow, import this news.opml file, courtesy of Wayne Price.

For more information

Useful RSS links from the government of Utah
http://gils.utah.gov/rss/

The RSS FAQ
http://www.voidstar.com/module.php?id=129&mod=book&op=feed

A list of RSS readers/clients
http://www.ourpla.net/cgi-bin/pikie.cgi?RssReaders

Microsoft-related RSS feeds, including MS developers
http://www.microsoft-watch.com/article2/0,4248,1097192,00.asp

The Userland RSS Feeds list
http://backend.userland.com/directory/167/feeds

Radio Community Server Top 100 RSS Feeds
http://radio.xmlstoragesystem.com/rcsPublic/rssHotlist

Reader Comments

Add a comment

Clarion Magazine

Topics

How to Display An Image In A ListBox Header (Part 1)

by Randy Rogers

Published 2003-06-23

Have you ever wished that you could place an image on a listbox header? In this article I will explain a technique (that is, a hack) that I use to place little triangles in the header, like those used in Microsoft Outlook to display a column's sort order. I will present two methods, and along the way expose something you may not realize about programming in the Windows environment.

The Clarion list control does not provide a mechanism for displaying images in the listbox header. If you try to place an image control or button on a list box header you will soon realize that the listbox takes priority on the screen, and it seems impossible to get an image to display there – they always end up being drawn underneath the listbox. The trick to solving this problem is getting the image to paint itself on top of the listbox header after the listbox header is redrawn. To make this happen you need the help of the Windows API `InvalidateRect` function.

`InvalidateRect` tells Windows a specific area on screen needs to be redrawn. As I'll demonstrate, you can use `InvalidateRect` to make an image appear on top of the list box, instead of hiding underneath.

So, lets get started. First create a new ABC application called DEMO.APP. You won't need a dictionary, and `Main` is fine for the first procedure's name.

Adding the API prototype

Now you need to add the prototype for `InvalidateRect` to the application. Go to **Global Embeds**, **Inside the Global Map**, and add the following:

```
MODULE('Win32 API')
```

```
  InvalidateRect(UNSIGNED hWnd, UNSIGNED lpRect, BOOL bErase),↵
     BOOL, RAW, PASCAL, PROC
END
```

The three parameters for the function call are:

`hWnd`	This is window handle of the window you want to work with. I'll get to this shortly.
`lpRect`	This is a pointer to a **RECT** structure. If you pass zero for this parameter then the entire window will be redrawn. You will pass zero (don't worry, it won't draw the whole Clarion window – I'll explain why later), so you don't need to worry about what a **RECT** structure looks like.
`bErase`	If this parameter is true and an update region for the window exists, a wm_erasebkgnd message will be sent to the window. You can use a false value for this parameter.

The Main procedure

Now you can create the Main procedure. Create a new procedure, and select **Window – Generic Window Handler** as the procedure template type.

On the procedure properties window, press the **Data** button and enter the following data elements:

```
ListBoxQueue              QUEUE,PRE(ListBoxQueue)
Field1                      STRING(20)
Field2                      STRING(20)
Field3                      STRING(20)
                          END
fField1Descending         BYTE
fField2Descending         BYTE(1)
fField3Descending         BYTE(1)
CurrentSortColumn         BYTE(1)
```

The `ListBoxQueue` will be used to populate the list box. The `fField`*`n`*`Descending` flags are used to keep track of the ascending/descending state of each column's sort order. You'll also need to keep track of the CurrentSortColumn so you'll know where to position the image.

In this initial methodology you will use a window timer event to cause the `InvalidateRect` code to get executed. In Part 2 of this article I will look at making the process more efficient by using window subclassing.

On the procedure properties window, press the **Window** button and create a window similar to the one shown in Figure 1. Make the frame resizable (to keep the example realistic) and set the window timer variable to 10. Populate the list control *without* a control template – you're going to use code to populate it with some test values. Use `?ListBox` for the use variable and the `ListBoxQueue` you defined earlier for the from variable for the list control. Add a vertical scroll bar so you can scroll the list. Right click on the list control and alert the `MouseLeft` key as shown in Figure 2. Use the **CloseButton Control Template** to add a **Close** button to the window.

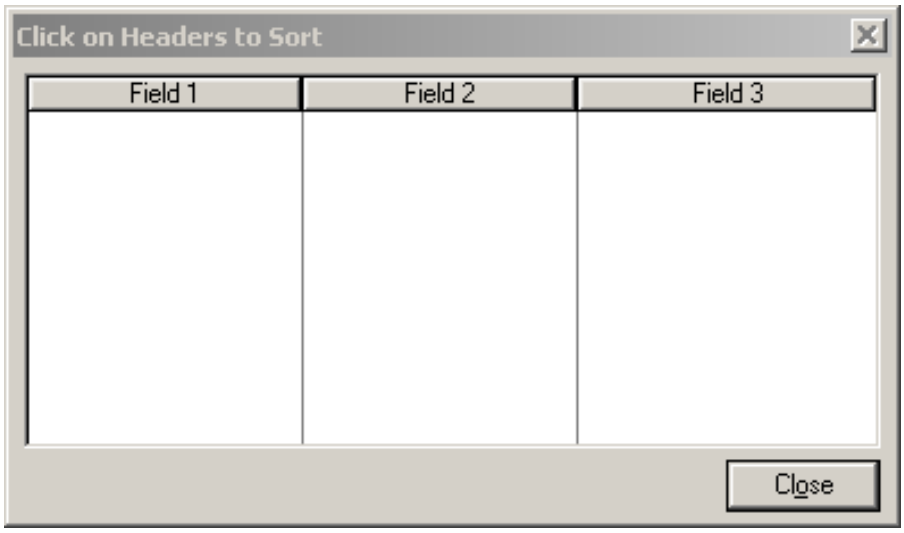

Figure 1. Window with a list control and Close button.

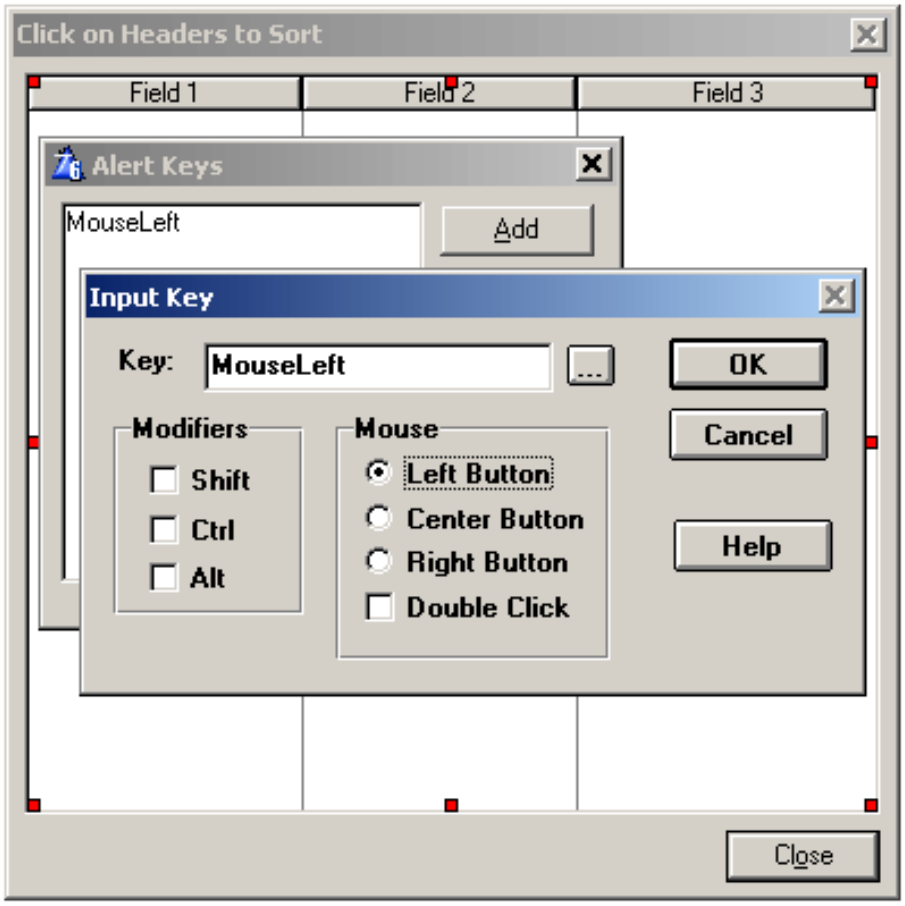

Figure 2. Alerting the MouseLeft key

Place a small, flat, transparent button on the window. You can place it anywhere on the window because you are going to control its position programmatically. I placed mine to show approximately where I wanted the image to initially appear. See Figure 3. Set the button's image property to ascending.gif and use `?SortImage` for the use variable.

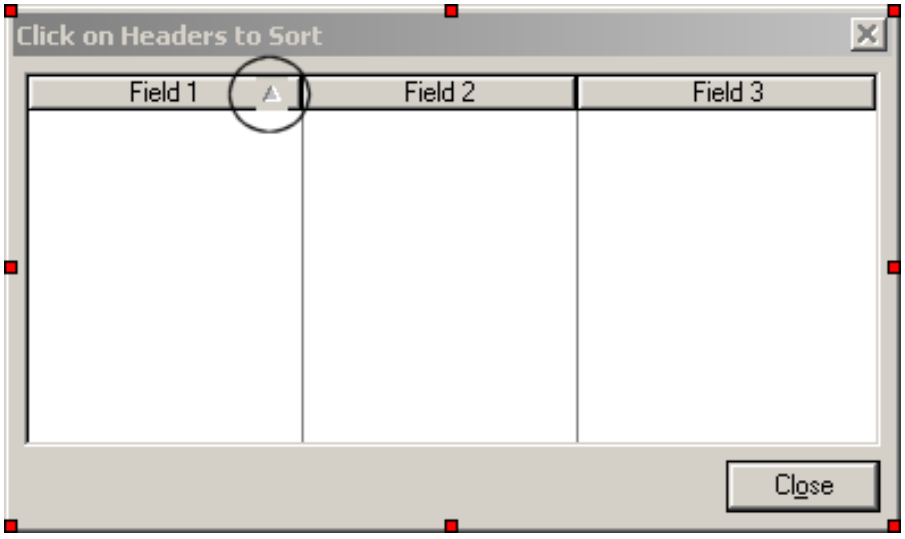

Figure 3. Adding the button with the icon image

Exit the window formatter.

Resizing settings

Press the Extensions button on the procedure properties window and add the **WindowResize** extension template. Select **Don't Alter Controls** for the **Resize Strategy** variable, and check **Restrict minimum window size**. Click the **Override Control Strategies** button and set the following control strategies:

`?ListBox` – Constant Right Border, Constant Bottom Border, Fix Left, Fix Top

`?CloseButto`n – Lock Width, Lock Height, Fix Right, Fix Bottom

`?SortImage` – Disable resizing for this control

Click on the **Embeds** button, and choose the **Local Objects, ABC Objects, Window Manager, Init, DATA** embed, priority 5000. Add the following:

```
I LONG,AUTO
J LONG,AUTO
```

In the **Local Objects, ABC Objects, Window Manager, Init, CODE** embed, priority 6500, place this code:

```
!This bit of code fills the ListBoxQueue with 100
! records containing random sample data.
LOOP I = 1 TO 100
  LOOP J = 1 TO 20
    ListBoxQueue.Field1[J] = CHR(RANDOM(VAL('A'),VAL('Z')))
    ListBoxQueue.Field2[J] = CHR(RANDOM(VAL('A'),VAL('Z')))
    ListBoxQueue.Field3[J] = CHR(RANDOM(VAL('A'),VAL('Z')))
  END
  ADD(ListBoxQueue)
END
SORT(ListBoxQueue,+ListBoxQueue.Field1)
```

In the same embed section, Priority 8030, add this code:

```
!Buffer the window for better redrawing
Window{PROP:Buffer} = 1
```

In the **Local Objects, ABC Objects, Window Manager, TakeWindowEvent, CODE** embed, priority 1300, add the call to `InvalidateRect`:

```
InvalidateRect(?SortImage{PROP:Handle},0,FALSE)
```

Remember when you set the timer attribute on the window? `TakeWindowEvent` responds to all window events, including timer events. Whenever the timer fires (ten times per second in this example), or any other window event occurs, the button image will be redrawn. The timer just provides a steady stream of events to keep the image refreshed because there is no event possted when the listbox is redrawn.

At the beginning of this article I said I would expose something about the windows programming environment that you might not realize. Notice where you are getting the window handle from for the first parameter of the `InvalidateRect` call… it's the icon button you placed on the window. *All the controls on your window are actually windows themselves.* When I discovered this a few years ago it was like the lights coming on; this realization opened the way for me to tackle problems like displaying images on the listbox header.

Because the button is itself a window, `InvalidateRect` tells only the button to redraw, and that's a small area. As I said, it's a hack, and there is a better way (as I'll explain next time) but it's still a fairly minor use of CPU cycles.

You should be able to compile and run the application now. It won't do much yet since you haven't added the code to handle sorting the listbox when the user clicks on the header, and you haven't dealt with the issues of column width changes or window resizing, but at least you should be able to see the image on the list box header.

Controlling image placement

When the user clicks on a column header, you want to display the image in that column header to show that it is the current column used for sorting. Additionally you need to indicate whether the column is sorted in ascending or descending order. You will want to create a routine to handle this since as the image needs to be positioned not only when the user clicks a column header but also when the columns are resized. Press the **Embeds** button on the procedure properties window and enter the following code snippet in the procedure routines section of the embed tree:

```
SetSortImagePosition    ROUTINE
  CASE CurrentSortColumn
  OF 1
    ?SortImage{PROP:Icon} = |
            CHOOSE(fField1Descending=TRUE,'~DESCENDING.GIF',|
            '~ASCENDING.GIF')
```

```
    ?SortImage{PROP:XPos} = ?ListBox{PROP:XPos}             |
                          + ?ListBox{PROPLIST:Width,1} - 14
  OF 2
    ?SortImage{PROP:Icon} = |
            CHOOSE(fField2Descending=TRUE,'~DESCENDING.GIF',|
            '~ASCENDING.GIF')
    ?SortImage{PROP:XPos} = ?ListBox{PROP:XPos}             |
                          + ?ListBox{PROPLIST:Width,1}      |
                          + ?ListBox{PROPLIST:Width,2} - 14
  OF 3
    ?SortImage{PROP:Icon} = |
            CHOOSE(fField3Descending=TRUE,'~DESCENDING.GIF',|
            '~ASCENDING.GIF')
    IF RECORDS(ListBoxQueue) <= ?ListBox{PROP:Items}
       ?SortImage{PROP:XPos} = ?ListBox{PROP:XPos} |
                             + ?ListBox{PROP:Width} - 14
    ELSE
       ?SortImage{PROP:XPos} = ?ListBox{PROP:XPos} |
                             + ?ListBox{PROP:Width} - 24
    END
  END
  ?SortImage{PROP:YPos} = ?ListBox{PROP:YPos}
  EXIT
```

This routine uses the `CurrentSortColumn` (set elsewhere) to help determine the image placement. For each column except the last, it calculates the sum of the widths of the columns up to and including the current column and then subtracts 14 DLUs to compensate for the width of the button, and keep it about five DLUs to the left of the column resize area.

The last column is a bit trickier. Here you want to position the image at the right of the column header as in the previous columns but, in the case of the last column, the physical column width can be greater than the column width reported by `PROP:Width`. Also there may or may not be a scroll bar visible, and you'll need to account for the width it takes when present.

The column flags are used to determine which image to display. Since the original button image is `ascending.gif` it is automatically included in the project, but you need to add `descending.gif` to the project tree to have it included. Select **Project|Properties** from the IDE menu and add `descending.gif` to the **Library, object, and resource files** node of the project tree.

Handling the header mouse click

To support browse sorting by clicking on the headers, add the following code to the **Control Events, ?ListBox, AlertKey** embed, priority 5000:

```
CASE KEYCODE()
OF MouseLeft       !The Alert Key
   IF ?ListBox{PROPLIST:MouseDownZone} = LISTZONE:HEADER
      IF CurrentSortColumn <> ?ListBox{PROPLIST:MouseDownField}
         EXECUTE CurrentSortColumn
           fField1Descending = TRUE
           fField2Descending = TRUE
           fField3Descending = TRUE
         END
      END
      CurrentSortColumn = ?ListBox{PROPLIST:MouseDownField}
      CASE CurrentSortColumn
      OF 1
         fField1Descending = BXOR(fField1Descending,1)
         IF fField1Descending
            SORT(ListBoxQueue,-ListBoxQueue.Field1)
         ELSE
            SORT(ListBoxQueue,+ListBoxQueue.Field1)
         END
      OF 2
         fField2Descending = BXOR(fField2Descending,1)
         IF fField2Descending
            SORT(ListBoxQueue,-ListBoxQueue.Field2)
         ELSE
            SORT(ListBoxQueue,+ListBoxQueue.Field2)
         END
      OF 3
         fField3Descending = BXOR(fField3Descending,1)
         IF fField3Descending
            SORT(ListBoxQueue,-ListBoxQueue.Field3)
         ELSE
            SORT(ListBoxQueue,+ListBoxQueue.Field3)
         END
      END
      DO SetSortImagePosition
   END
END
```

This code checks for the alert key and determines if the listbox header was clicked. If it was, it checks to see if the user is changing columns and, if a new column has been selected, flags the current column as descending so that the next time it is clicked it will default to ascending sort order. Next it sets the `CurrentSortColumn`, flips the appropriate `fFieldnDescending` flag and sorts the `ListBoxQueue` accordingly. Finally it calls the `SetSortImagePosition` routine to set the image position.

In the **Control Events, ?ListBox, PreAlertKey** embed, priority 4499, add this code:

```
IF KEYCODE() <> MouseLeft
   CYCLE
ELSIF ?ListBox{PROPLIST:MouseDownZone} <> LISTZONE:HEADER
```

```
   CYCLE
END
```

You need to take a bit of extra care if you want to use a single mouse click on a column header becaue the `MouseLeft` key also controls the resizing of the columns. Without this bit of code the columns won't resize. If the keycode is not the alert key, the code cycles to allow normal processing of the keycode. If the keycode is the alert key and it is not in the listbox header zone, the code also cycles to allow normal processing. Otherwise, the code does nothing and the alert key embed code will get executed.

Handling Resizing

To handle column resizing you need to call the `SetSortImagePosition` routine at the appropriate point. Place the routine call in the **Control Events, ?ListBox, ColumnResize** embed point, priority 5000:

```
DO SetSortImagePosition
```

To handle window resizing, place the following code in the **Local Objects, ABC Objects, Window Resizer, Resize, CODE** embed point, priority 5001:

```
POST(EVENT:ColumnResize,?ListBox)
```

When the listbox resizes, the column widths may change, particularly the rightmost column, so you handle that by posting the `ColumnResize` event to the listbox.

Summary

In this article I have tackled the problem of displaying an image in the list box header by using a window timer event to trigger the `InvalidateRect` windows API call which provides the needed magic. This techniques works, but is not very efficient processing-wise and may not mesh too well with applications that are already relying on the timer event for other purposes. In Part 2 of this article I will show you how to improve the technique by using window subclassing on Clarion controls, now exposed as windows in their own right!

Download the source

Randy Rogers is a data processing professional with over 35 years of experience in a wide variety of industries

including accounting, municipal government, insurance, printing, and pharmacoeconomics. He has a degree in Mathematics from Florida State University and is the president of Keystone Computer Resources. Randy is the author of ClassViewer, a utility for browsing the Clarion class hierarchies. He is also the creator of NetTools, Queue Edit-in-Place, and Screen Capture Tools for Clarion application developers.

Reader Comments

Add a comment

Randy, Excellent, I've been looking for this solution...
I second Steffen's opinion... Can't wait to read the...

Clarion Magazine

A Tree In A Page Loaded Browse: Update And Delete Logic

by Ronald van Raaphorst

Published 2003-06-23

In my previous article I showed that the Clarion RelTree template is not designed to contain a flexible number of levels in a tree, and database design is difficult when a single file links to multiple levels. I then presented another solution, which instead uses one file for all levels, and links the levels by the contents of the `SeqNo` field. To display the tree, I use the standard (page loaded) BrowseClass, and use the `SeqNo` field to determine the order and level of each record.

In that article I also discussed how to add new records to the tree browse. In this article I will examine record actions more closely.

Record actions

Part 1 dealt mostly with how to displaying the records in tree format; now it's time to look at the Add, Change, and Delete actions in more detail. This will be a discussion of the coding requirements; in Part 3 I'll look at the implementation.

> **NOTE:** In Part 1 I talked about Organization, Department and Subdepartment records. For simplicity, in this article I will call Organization records (or Level 1 records) root records, and records with a higher level number child records. Note that this has nothing to do with file relations, as the root and the child records are stored in the same file. I'll also be talking about child-to-child records, which are what I previously called subdepartments.

Now lets have a closer look at the three record actions:

Inserting records

If there are no records at all when a record is inserted, the first record is primed and the SeqNo field is set to `'0001'`. If this record is selected and a new record is inserted, the new record automatically becomes a child of the first record. So how do you insert new records which are not children of the first record? That's currently not possible. So, first, you have to find out if the user wants to insert a child record or insert a new root record.

Some possible options are:

- Add a new button for adding a root record
- Use a popup or dropdown menu
- Left click on the Insert button to add a child, right click on the Insert button to add a new root record (not standard Windows behavior).
- Take action based on the currently selected record. If the current record level is 1 or if no record is selected, ask the question; if the record level is 2 or higher, add a child record.

As I hate to add a button for every new function, I'd like to reuse the Insert button, and extend its functionality a little. I therefore will use one Insert button with double functionality, as shown in Fig. 1. The button text changes depending on what the user selects from the dropdown menu.

Figure 1. Insert button with extra functionality

Once you pick an option, you use that option until you select the other one.

For simplicity, records are always inserted at the end of the currently selected level. See Part 1 for the code to prime a `SeqNo` field according to a parent record.

Changing records

To maintain file integrity, changes in the `SeqNo` field should be monitored and any changes cascaded. Typically, an end-user can't see or change this field directly, so there are no problems here.

On the other hand, maybe you want to give the end-user the ability to move a child record to another parent record. This way, you can move a department to another department.

Take a look at what happens if you assign a record to another (root or child) record. In comparison to the previous article, I added Budweiser as a third root record. Assume that Heineken is bought by Grolsch, so Heineken ('`0001`') becomes a child record of Grolsch:

Before assignment		After assignment	
Name	**SeqNo**	**Name**	**SeqNo**
Heineken	0001	Grolsch	0002
Sales	0001.0001	Sales	0002.0001
Production	0001.0002	Production	0002.0002
Grolsch	0002	R&D	0002.0003
Sales	0002.0001	Heineken	0002.0004
Production	0002.0002	Sales	0002.0004.0001
R&D	0002.0003	Production	0002.0004.0002
Budweiser	0003	Budweiser	0003

Table 1. Assigning a record to another record

Because Grolsch is the new parent record, which is a level 1 record, I locate the last child on the next (second) level, and add one to its sequence. The result is `'0002.0004'`. This is basically the same autonumbering action as described in my previous article for inserting a new child record. As I don't want to write code twice, and because I'm some kind of an Architecture Astronaut, I'll just have to reuse some of that code!

After the SeqNo of the Heineken (`'0001'`) record is changed to `'0002.0004'`, you can replace the `'0001'` part of the `SeqNo` field of all records where the `SeqNo` starts with `'0001'` with `'0002.0004'`. So the `SeqNo` field of the Heineken Sales department (`'0001.0001'`) becomes `'0002.0004.0001'` etc.

In other words, prototyping the process gives:

```
SeqNo_Old = '0001'
SeqNo_New = '0002.0004'
LOOP All child records ( SUB(SeqNo, 1, LEN(SeqNo_Old)) = SeqNo_Old )
  SeqNo = SeqNo_New & SUB(SeqNo, LEN(SeqNo_Old)+1, LEN(SeqNo))
  Do the same for child records of this record
```

```
END
```

Or, filled in for the child sales ('0001.0001'):

```
LOOP All child records (SUB('0001.0001', 1, LEN('0001')) = '0001'
  SeqNo = '0002.0004' & SUB('0001.0001', LEN('0001')+1, LEN('0001')) equals
  '0002.0004' & SUB('0001.0001', 4 + 1, 5) equals
  '0002.0004' & '.0001'
  Do the same for child records of 'Sales'
END
```

Notice that this will work even if Heineken was moved to Grolsch > R&D. In this case, `SeqNo_Old = '0001'` and `SeqNo_New = '0002.0003.0001'.`

It would be very nice to drag and drop records, and thus link them to another record. As I first have yet to dive into the technical implementation, I can't promise this functionality yet.

You also may want to give the end-user the ability to move child records up and down. As this functionality typically will take place in the browse window, you need to give the end-user access to it. I will add Move Up and Move Down to the Browse's popup menu..

Moving a record up or down within a level means switching the last four characters, for these characters determine the display order. The length of the SeqNo field (and thus the level displayed) stays the same.

Deleting records

If you delete a record, you have a designer's choice to either delete all child records, or to move all child records one level up. This way, you can delete a level, but keep the children This I would ask the user every time, because it also gives the user the chance to cancel the action.

What are the actions you must take if you want to delete all child records? Assume you want to delete the `'0001'` (Heineken) record and its children. Table 2 shows the records before and after the delete action.

Before delete action		After delete action	
Name	**SeqNo**	**Name**	**SeqNo**
Heineken	0001	Grolsch	0002
Sales	0001.0001	Sales	0002.0001

Production	0001.0002	Production	0002.0002
Grolsch	0002	R&D	0002.0003
Sales	0002.0001		
Production	0002.0002		
R&D	0002.0003		

Table 2. Deleting a root record

The 'source' pseudocode can be written as:

```
SeqNo_Old = '0001'
LOOP All child records ( SUB(SeqNo, 1, LEN(SeqNo_Old)) = SeqNo_Old )
Delete record
END
```

It’s a little harder if you only want to delete the root record '0001', but keep the child records, as shown in Table 3.

Before delete action		After delete action	
Name	**SeqNo**	**Name**	**SeqNo**
Heineken	0001	Grolsch	0002
Sales	0001.0001	Sales	0002.0001
Production	0001.0002	Production	0002.0002
Grolsch	0002	R&D	0002.0003
Sales	0002.0001	Budweiser	0003
Production	0002.0002	Sales	0004
R&D	0002.0003	Production	0005
Budweiser	0003		

Table 3. Delete a root record but keep its children

The Heineken Sales and Production records have moved up a level from 2 to 1. But you can’t simply remove the first four characters, because the `SeqNo` field is unique. So you need to move the child records up a level, then autonumber them on that level, and update all of that record’s children.

To visualize this, take the result of Table 1, where Heineken is a child of Grolsch, and imagine you still want to delete the Heineken record. Table 4 shows the records before and after the delete action.

Before delete action		After delete action	
Name	**SeqNo**	**Name**	**SeqNo**
Grolsch	0002	Grolsch	0002
Sales	0002.0001	Sales	0002.0001
Production	0002.0002	Production	0002.0002
R&D	0002.0003	R&D	0002.0003
Heineken	0002.0004	Sales	0002.0004
Sales	0002.0004.0001	Production	0002.0005
Production	0002.0004.0002	Budweiser	0003
Budweiser	0003		

Table 4. Delete a child record and keep its children

In this example, the `SeqNo` of the Heineken Sales record is changed from `'0002.0004.0001'` to `'0002.0004'`. In this case, it looks like I have removed only the last characters, but that is a coincidence and may be a bad example.

Here's some source for deleting a record.

```
SeqNo_Deleted = '0002.0004'
SeqNo_Parent = SUB(SeqNo_Deleted,1,LEN(SeqNo_Deleted)- |
CHOOSE(LEN(SeqNo_Deleted)=4, 4, 5))
LOOP All child records ( SUB(SeqNo, 1, LEN(SeqNo_Deleted)) = SeqNo_Deleted)
Autonumber, based on parent (SeqNo_Parent), and add this record
Assign child records to new SeqNo
END
```

And verify the Parent calculation:

```
SeqNo_Deleted = '0002.0004'
SeqNo_Parent = SUB('0002.0004',1,LEN('0002.0004')- |
CHOOSE(LEN('0002.0004')=4, 4, 5)) equals
SUB('0002.0004', 1, 9 - CHOOSE(9 = 4, 4, 5)) equals
SUB('0002.0004', 1, 9 - 5) equals
'0002'
```

Notice the `CHOOSE` function in the `SeqNo_Parent` calculation. If you are deleting a root record, its length equals four characters, and the result is zero. Thus, `SeqNo_Parent` becomes `''`.

If you are deleting a child record, the condition of the `CHOOSE` statement equals false, the result of the `CHOOSE` statement equals five, and thus the last five characters (incl. numbers and separating period) are removed.

Conclusion

What looked like a simple and elegant solution for a tree in a page loaded browse box, becomes a little more complex if you take a look at the record actions you will want to do.

Because of the recursive nature of the actions (loop (child)records and their children), you can't simply write a template and add some code to embeds. Though it is possible to code recursive loops as a single loop and mess with variables, I like the clean code of a single class method calling itself. So my next article in this series will be about designing and implementing a class to handle these requirements.

Ronald van Raaphorst studied Chemical Engineering at the University of Enschede (UT), the Netherlands. But he found programming was more fun than designing a chemical plant, and when a roommate asked him to help start a software company, he found the choice easy to make. Ronald has used Clarion since 1994, beginning with Clarion 3 for DOS. Compad Software, which he co-owns, sells software to a small group of bakeries, he spends a considerable amount of time on the phone helping users, finding (and creating) new bugs, writing manuals, and of course programs. Ronald is in charge of developing Compad's products, and his colleague is on the road selling.

Reader Comments

Add a comment

Clarion Magazine

Topics

How to Display An Image In A ListBox Header (Part 2)

by Randy Rogers

Published 2003-06-25

In Part 1 of this article I showed you how easy it is to display an image on a listbox header. The trick is to use the `InvalidateRect` windows API function to get the image to redraw. Because there is no window event posted when the listbox is redrawn, I used a timer event to trigger a steady stream of redraws. Actually I placed the `InvalidateRect` API call at the top of the `WindowManager.TakeEvent` method to have the image redraw on any event (not just the timer event). As you might imagine this is not very efficient, since the image is redrawn not just on timer events, but also on every other window event that the Clarion runtime library passes to the application. The image really only needs to be redrawn when the listbox is redrawn, and that's the subject of this second part.

The trick to drawing images on list box headers efficiently is to know when the listbox itself is redrawn. You'll recall that in Part 1, I "exposed" Clarion controls for what what they really are – windows in their own right. And since the listbox is just another window, the more generic question is how to determine when a window is redrawn.

When a portion of the application window needs to be redrawn, the operating system sends a `wm_paint` message to the window. A window receives this message through its `WindowProc` function. The Clarion runtime library provides the `WindowProc` processing for your application windows, and also for the individual control windows, and handles the `wm_paint` messages somewhere inside the `ACCEPT` loop's black box.. Normally that's ideal, only this time you want to handle some of the repainting yourself. The trick here will be to replace the `WindowProc` with your own. Fortunately, Clarion makes this really easy via the `prop:wndproc` runtime property.

Getting started

Start by opening the demo.app you created in Part 1 of the article. You will first want to remove the timer from the window and remove or comment out the `InvalidateRect` call at the top of the `WindowManager.TakeEvent` method. You can leave the rest of the code because you will still need it. Additionally, you will need to use the `CallWindowProc` window API function to make this work. Go to **Global Embeds**, **Inside the Global Map**, and add the following line above the `InvalidateRect` declaration:

```
CallWindowProc(UNSIGNED lpPrevWndFunc, UNSIGNED hWnd, UNSIGNED Msg,UNSIGNED
  wParam, LONG lParam),LONG,PASCAL,NAME('CallWindowProcA')
```

The five parameters for the function call are as follows:

`lpPrevWndFunc`	A pointer to the previous window procedure. You will use `prop:WndProc` to get the value for this parameter.
`hWnd`	This is the handle of the window procedure to receive the message.
`Msg`	Specifies the message. You will just pass on the one you receive by calling `CallWindowProc` after your code finishes.
`wParam`	This parameter specifies additional message-specific information. You will just pass on the value you receive.
`lParam`	This parameter specifies additional message-specific information. Again, you will just pass on the value you receive.

Saving the WindowProc address

When you create a `WindowProc` function to handle some specialized window painting (or other) code, you don't completely replace the stock `WindowProc` code. Instead, you tell Windows to call your `WindowProc` function, and then your `WindowProc` function calls the original. So that means you have to create a variable to hold the original `WindowProc` address

On the procedure properties window, press the **Data** button and add the following

data element:

```
Save:ListBox::WindowProc LONG,STATIC
```

NOTE: *Make sure you declare the variable's storage class as* `STATIC`. If you do not do this, your application will GPF. You can specify a variable's **Storage Class** on the **Attributes** tab of the Property dialog for the variable. See Figure 1.

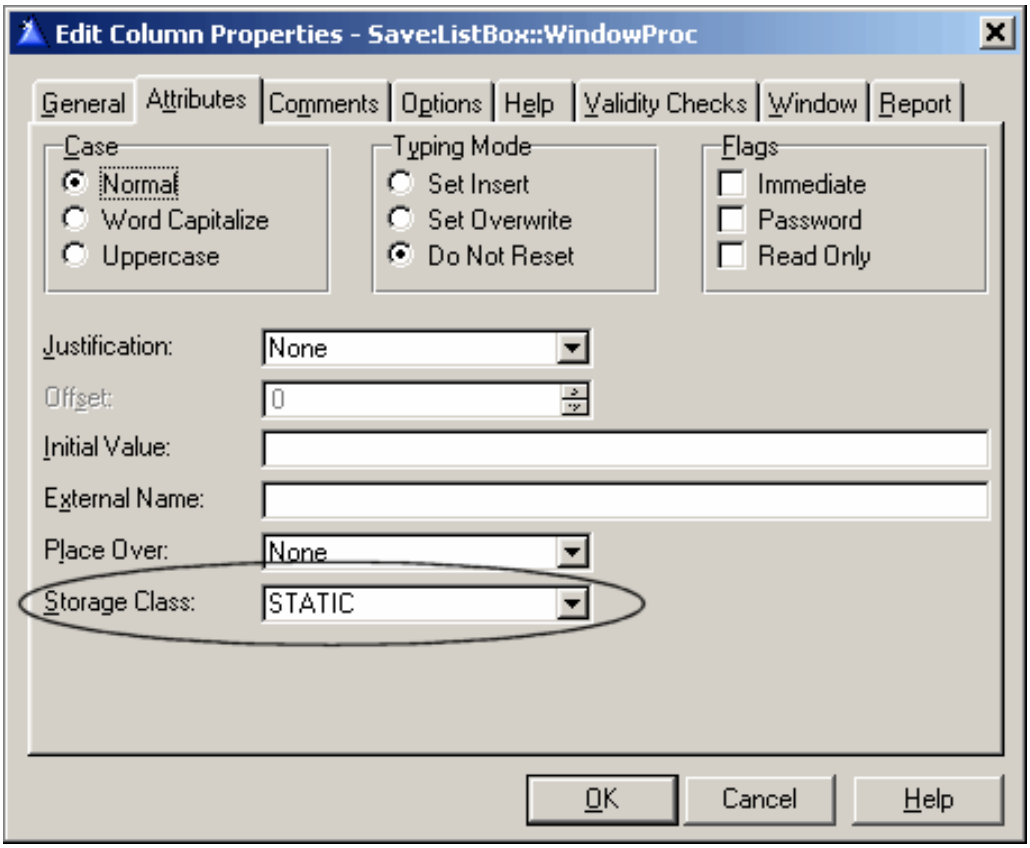

Figure 1 Setting the variable Storage Class

Creating a WindowProc procedure

Press the **Embeds** button on the procedure properties screen and add the following source code to **Local Procedures,** priority 4000:

```
ListBox::WindowProc PROCEDURE(UNSIGNED hWnd, UNSIGNED wMsg,|
                      UNSIGNED wParam, LONG lParam)
WM_PAINT    EQUATE(0000Fh)
  CODE
   CASE wMsg
   OF WM_PAINT
      POST(EVENT:USER)
   END
   RETURN(CallWindowProc(Save:ListBox::WindowProc,hWnd,|
      wMsg,wParam,lParam))
```

This procedure will become the new WindowProc for the list control window. It

looks for the wm_paint message and posts an event:user event to the application window when one arrives. You will use this event to invoke the InvalidateRect API function. The procedure then passes control on to the original WindowProc so that the Clarion runtime can process the message as it normally would.

The Clarion IDE does not automatically add procedures declared in the LocalProcedures embed to the module map, so you will need to either place the declarations in the map using the module embed point, or have a template do it for you. Here's a small template that will add ListBox::WindowProc to the module map.

Create a file named 'demo.tpl' containing the following statements:

```
#TEMPLATE(DEMO,'Clarion Magazine'),FAMILY('ABC')
#EXTENSION(Custom_Module,'Add Custom Module Procedure'),PROCEDURE
#AT(%CustomModuleDeclarations)
#ADD(%CustomModuleMapModule,'CURRENT MODULE')
#SET(%ValueConstruct,'ListBox::WindowProc')
#ADD(%CustomModuleMapProcedure,%ValueConstruct)
#SET(%CustomModuleMapProcedurePrototype,↵
 '(UNSIGNED,UNSIGNED,UNSIGNED,LONG),LONG,PASCAL')
#ENDAT
```

Save the file with a name of your choice (say, "demo.tpl") in your Clarion template directory, save and close your application, register the template, and then reopen your demo app. Press the **Extensions** button on the procedure properties window and add the template to the `Main` procedure.

Subclassing the listbox WindowProc

Now it's time to install the event handling procedure for the list box control. This process is also called subclassing. In the **Local Objects, ABC Objects, Window Manager, Init, CODE** embed, priority 8040, place this code:

```
!Subclass the ListBox WindowProc
Save:ListBox::WindowProc = ?ListBox{PROP:WndProc}
?ListBox{PROP:WndProc} = ADDRESS(ListBox::WindowProc)
```

This code saves the current `WindowProc` in the `Save:ListBox::WindowProc` variable you created earlier. This variable is used in the `CallWindowProc` function you coded in the `ListBox::WindowProc` procedure.

Before calling the `WindowManager.Kill` method you will want to "un-subclass" the listbox `WindowProc` and restore it to the original. In the Local

Objects, ABC Objects, Window Manager, Kill, CODE embed, priority 4500, place this code:

```
!Restore the original ListBox WindProc
?ListBox{PROP:WndProc} = Save:ListBox::WindowProc
```

Handling EVENT:USER

All that remains is to make the event:user posted by your `WindProc` trigger the call to `InvalidateRect`, which forces the redisplay of the image on the listbox header. In the **Local Objects, ABC Objects, Window Manager, Init, CODE** embed, priority 5001, place this code:

```
IF EVENT() = EVENT:USER
   InvalidateRect(?SortImage{PROP:handle},0,FALSE)
END
```

You should now be able to compile and run your revised application, and each time the listbox is redrawn, the header image will also be redrawn.

Summary

In this part of the article I have shown how to exploit the fact that Clarion controls are actually individual windows. By subclassing the listbox control I was able to determine when the control gets repainted. This allowed me to remove the dependence on the timer event to improve the efficiency of the hack. I also provided a simple template to facilitate the addition of the new `WindowProc`. The template could easily be modified to prompt for a procedure name and prototype, and could be further expanded to automatically generate much of the code presented in the two parts of this article. I leave that as an exercise for the reader.

Download the source

Randy Rogers is a data processing professional with over 35 years of experience in a wide variety of industries including accounting, municipal government, insurance, printing, and pharmacoeconomics. He has a degree in Mathematics from Florida State University and is the president of Keystone Computer Resources. Randy is the author of ClassViewer, a utility for browsing the Clarion class hierarchies. He is also the creator of NetTools, Queue Edit-in-Place, and Screen Capture Tools for Clarion application developers.

Reader Comments

Add a comment

Clarion Magazine

A String To CString Converter

by David Harms

Published 2003-06-26

I've been playing around with the Clarion 6 wrapper for the CenterPoint DOM parser, and one thing I noticed right away was that a lot of methods expected `CSTRING` parameters, passed by address. That's fine, except that in many cases I wanted to simply pass in a string literal, without having to put the text into a `CSTRING` variable first, i.e. `SomeClass.SomeMethod('SomeText')`. This would be easy if Clarion permitted `CSTRING` literals, but it doesn't, at least not as of C6 EA4-5.

At first, I simply set up a function that allocated a `CSTRING` using `NEW()`, and returned that. I guess I've been spending too much time using a certain garbage-collected language (which shall remain nameless), because it was quickly pointed out to me that I was allocating memory, but not `DISPOSE`ing of it, thereby creating a memory leak. After a lengthy discussion in the newsgroups with Jim Gambon, to which Jeff Slarve and Dennis Evans also contributed, I ended up creating a small class with a method that receives a `STRING` parameter (by value) and returns a CSTRING version of the data. This class hangs onto the `CSTRING`s and `DISPOSE`s of them when they're no longer needed.

Now, instead of this:

```
text       CSTRING(20)
...
  CODE
  text = 'Some Text'
  SomeClass.SomeMethod(text)
```

I can declare an instance of my class, give it the label s, and convert a string literal on the fly:

```
SomeClass.SomeMethod(s.GetCString('Some Text'))
```

I can also use the shortcut method called c, which calls GetCString:

```
SomeClass.SomeMethod(s.c('Some Text'))
```

For an example of this usage, see my article in this issue on the C6 DOM parser.

The code

Here's the declaration for the class (ccics.inc):

```
!ABCIncludeFile

OMIT('_EndOfInclude_',_cciCStringFactoryPresent_)
_cciCStringFactoryPresent_ EQUATE(1)


CStringQ        queue,type
cs                  &CString
                end

cciCStringFactory  CLASS(),TYPE,MODULE('ccics.clw'),|
                      LINK('ccics.clw',_ABCLinkMode_),|
                      DLL(_ABCDllMode_)
q                    &CSTringQ
Construct             procedure
Destruct              procedure
GetCString            procedure(String s),*CString
C                     procedure(String s),*CString
                   end

_EndOfInclude_
```

The only item of note in the declaration is that there is a typed queue (`CStringQ`) of `CSTRING` reference variables, and the class contains a reference of that queue type. The class can then create the required queue at startup.

Here's the implementation (ccics.clw):

```
MEMBER


  MAP
  .
  INCLUDE('ccics.inc')

cciCStringFactory.Construct          procedure
    code
    self.q &= new(CStringQ)
```

```
cciCStringFactory.Destruct          procedure
x       long
    code
    loop x = 1 to records(self.q)
        get(self.q,x)
        dispose(self.q.cs)
    end
    free(self.q)
    dispose(self.q)


cciCStringFactory.GetCString        procedure(String s)

    code
    if records(self.q) > 10
        get(self.q,1)
        dispose(self.q.cs)
        delete(self.q)
    end
    clear(self.q)
    self.q.cs &= new cstring(len(clip(s)) + 1)
    add(self.q)
    self.q.cs = s
    return self.q.cs


cciCStringFactory.c                 procedure(String s)
    code
    return self.GetCString(s)
```

The Constructor and Destructor simply take care of initializing and cleaning up the queue of `CSTRING`s. The `GetCString` method does all the work of creating a new `CSTRING`, using the length of the passed string plus one. Remember that `CSTRINGs` are terminated by a null character and so are always at least one character longer than the data. `GetCString` also trims the first record from the queue if there are more than 10 CSTRINGs present so that memory usage won't get out of hand if the class is used extensively. The `c` method is just a shorthand for `GetCString`.

Larry Sand reminded me that Alexey Solovjev, SoftVelocity's guru of the compiler (and much else), has stated in the newsgroups that `*CSTRING` is not a correct return type, but it is allowed by the compiler. Make of that what you will.

Using the class

This class is not thread safe, which means that you should only use an instance of it in a single thread. For instance, if you declare an instance in a procedure data section as follows, you can use the class without trouble:

```
s cciCStringFactory
```

The class will be allocated memory when the procedure loads, the constructor will fire and create the queue, and when the procedure terminates the destructor will clean up anything left in the queue.

You should *not* declare this class globally, however. If you have multiple threads calling the `GetCString` method at the same time, the queue cleanup code could be compromised, as could the reference to the new `CSTRING`.

Summary

At present, there is no easy way to pass a `CSTRING` literal to a function/method in Clarion. You either have to create a `CSTRING` variable, set its value, and pass the variable, or you need to use a utility class like `cciCStringFactory`. Although I created this class to make it easier to call the C6 DOM interface methods, it should work fine with Clarion 5.x as well.

Download the source

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of Developing Clarion for Windows Applications, published by SAMS (1995). His most recent book is JSP, Servlets, and MySQL, published by HungryMinds Inc. (2001).

Reader Comments

Add a comment

Why not: SomeObject.SomeMethod('Some Text<0>')
Charlie, Even with the <0> it's still a string literal,...
Looks like this class already exists in the xpcml.inc...
Gordon, Son of a gun - I should have noticed that....
I would agree it should be inhanced to include a queue...

Clarion Magazine

Creating XML Files With The Clarion 6 DOM Parser

by David Harms

Published 2003-06-26

Last month I wrote an article on XML for Clarion developers, and followed that up with a two part article showing how to create an RSS feed with the Clarion 6 (C6) XMLGenerator class. I chose RSS as an example of XML because RSS is one of the simpler XML specifications you're likely to encounter, and I chose the all-Clarion XMLGenerator class for its ease of use, and because it is already used in the C6 XML export functionality. But there is another, more powerful way to create XML files in Clarion, and that's using the CenterPoint XML library and wrapper classes that are part of C6.

In this article I'll show how to use the C6 DOM classes to recreate one of the CenterPoint DOM examples, and then I'll use DOM to create the same RSS XML file from my previous RSS article.

The CenterPoint parsers

C6 ships with a LIB version of the open source CenterPoint C++ XML class library, which includes SAX and DOM parsers. Licensing for the CenterPoint library is similar to the terms of the Netscape Public License – the general idea of the license is to make it possible for developers to include this code in their applications without causing the entire application to become open source. The CenterPoint code is derived from expat, the XML Parser Toolkit.

Clarion 6 also includes wrapper classes for both the SAX and DOM parsers, but as I'm mainly concerned with writing XML documents I've only explored the DOM side so far (and only writing, not reading, XML documents).

There is, as of this writing, no official C6 documentation for the CenterPoint wrappers, but there is a discussion forum for the C++ classes, and I also found some helpful examples in the source download.

The wrapper classes

You can find the CenterPoint wrapper classes in cpxml.inc and cpxml.clw. These aren't the easiest Clarion source files to read as they declare a lot of interfaces that are implemented in the C++ library, so the syntax is sometimes a bit foreign. I've been using the Keystone Class Viewer to help me find my way, as shown in Figure 1.

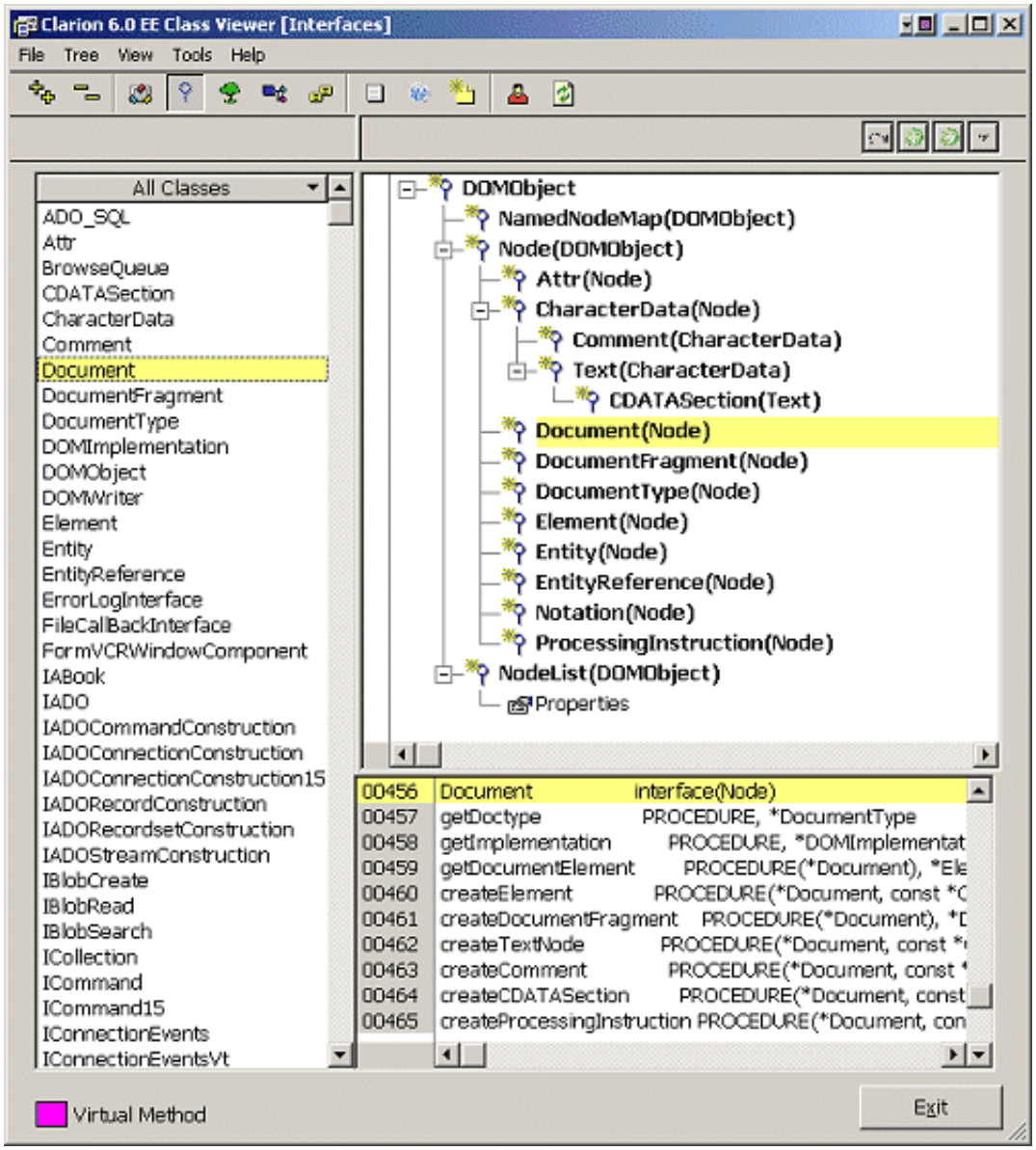

Figure 1. The Keystone Class Viewer showing the DOM Interface hierarchy

The basic steps to creating a document with C6 DOM are as follows:

- Use a library function to create a DOM implementation (an object which implements the `DomImplementation` interface)
- Call the DOM implementation object's `CreateDocument` method to get a `Document` object.
- Create various `Element` objects (i.e. tags), give them attributes (text, CDATA), and add them to the document. Elements can have child elements

- Use a library function to create an object which implements the `DOMWriter` interface.
- Use the `DomWriter` to write the XML document to disk

The CenterPoint XML source includes a samples\src directory, which contains, among other things a C++ file called WriterTest.cpp. I won't reproduce all the code here, just the block that actually writes the file. If you're not used to C++, you can render this code somewhat more readable by mentally replacing `::` and `->` with the dot operator (`.`) – it will no longer be valid C++, of course, but it will read more like Clarion.

```
string xhtmlns = "http://www.w3.org/1999/xhtml";
CSL::XML::DOMImplementation* pDomImpl =
   CSL::XML::DOMImplementation::getInstance();
CSL::XML::Document* pDoc = pDomImpl->createDocument(xhtmlns,
   "html", pDomImpl->createDocumentType("", "
   html", "xhtml-sample-1.dtd"));
CSL::XML::Element* pDocElem = pDoc->getDocumentElement();
CSL::XML::Comment* pCommentElem =
   pDoc->createComment("CenterPoint/XML Sample");
pDoc->insertBefore(pCommentElem, pDocElem)->release();
CSL::XML::Element* pHeadElem = pDoc->createElementNS(xhtmlns, "head");
CSL::XML::Element* pTitleElem = pDoc->createElementNS(xhtmlns, "title");
pTitleElem->appendChild(pDoc->createTextNode("XHTML Sample"))->release();
pHeadElem->appendChild(pTitleElem)->release();
pDocElem->appendChild(pHeadElem)->release();
CSL::XML::Element* pBodyElem = pDoc->createElementNS(xhtmlns, "body");
CSL::XML::Element* pParaElem = pDoc->createElementNS(xhtmlns, "p");
pParaElem->setAttribute("align", "center");
pParaElem->appendChild(pDoc->createTextNode(
   "Created with CenterPoint/XML."))->release();
pBodyElem->appendChild(pParaElem)->release();
pDocElem->appendChild(pBodyElem)->release();
CSL::XML::DOMWriter domWriter;
// enable pretty-printing, no XML declaration
domWriter.setFormat(CSL::XML::DOMWriter::REFORMATTED
   | CSL::XML::DOMWriter::CANONICAL);
domWriter.writeNode("test.xhtml", pDoc);
domWriter.writeNode(&cout, pDoc);
pDoc->release();
```

This code creates a small XML document that also happens to be valid HTML, and that looks like this:

```
<!--CenterPoint DOM example-->
<html xmlns="http://www.w3.org/1999/xhtml">
        <head>
                <title>XHTML Sample</title>
        </head>
        <body>
                <p align="center">Created with CenterPoint/XML.</p>
        </body>
</html>
```

It's actually quite easy to recreate this code in Clarion, as the C6 wrapper classes

maintain the CenterPoint/DOM naming convention. One thing you will need is a number of reference variables, as follows:

```
DOMImpl      &DomImplementation
docType      &DocumentType
prevNode     &Node
pDoc         &Document
pDocElem     &Element
pBodyElem    &Element
pParaElem    &Element
pCommentElem &Comment
pTitleElem   &Element
pHeadElem    &Element
pText        &Text
writer       &DomWriter
```

These are all actually references to interfaces, not to classes. Really it makes no difference to you – the objects themselves are created in the XML library, and it doesn't matter to you want actual object is returned, as long as it implements the methods in the interface signature. Also in the declaraction I have one cstring constant (because it gets used a few times):

```
xhtmlns      cstring('http://www.w3.org/1999/xhtml')
```

And finally I have a instance of a class called cciStringFactory, cryptically called s.

```
s            cciCStringFactory
```

cciStringFactory and CString literals

Getting the DOM code to work wasn't that hard – the biggest difficulty was that many of the DOM interface methods take parameters of type `CSTRING`. That is, they expect `CSTRING`s passed by address. If all of the data I wanted to pass happened to be in `CSTRING`s already, that would be fine, but in my testing I had a lot of string literals. I could not call a method this way:

```
pCommentElem &= pDoc.CreateComment('CenterPoint DOM example')
```

because the compiler sees any string literal as a `STRING`, not a `CSTRING`. I had to find a way of converting a `STRING` to a `CSTRING`. After a discussion in the newsgroups, to which Jim Gambon and Jeff Slarve contributed valuable information, I wrote a class that dynamically creates CSTRINGs from STRINGs as needed, and garbage collects them when done. As a result, after declaring an instance of `cciCStringFactory` called `s`, and given that `cciStringFactory` has a method called `c` (which is just a shorthand for a call to the `GetCString` method), I can now pass in my `CSTRINGs` this way:

```
pCommentElem &= pDoc.CreateComment(s.c('CenterPoint DOM example'))
```

For more on `cciCStringFactory` see the article in this issue. On to the Clarion

version of the CenterPoint example!

Creating the DOMImplementation

As I mentioned, the DOM wrapper consists mainly of interfaces. That means that when you're creating a base object such as DOMImplementation, you don't do it by declaring a DOM class instance, but by calling a library function that returns an object that implements the DOMImplementation interface.

```
!string xhtmlns = "http://www.w3.org/1999/xhtml";
!CSL::XML::DOMImplementation* pDomImpl =
!   CSL::XML::DOMImplementation::getInstance();
DomImpl &= CreateDomImplementation()
```

`DOMImplementation` is the basic object you need for any DOM work – think of it as a factory that provides you with the components you need to create your document. And speaking of documents, it's where you get that too.

There are two steps to creating a document. One is to get a `DocumentType` object, which defines the document's characteristics. For simple documents this object can actually be null, as it will be in the following RSS example. To create the document, you pass the document's namespace URI (if any), the name of the root tag, and the `DocumentType` to the `DomImplementation`'s `CreateDocument` method.

```
!CSL::XML::Document* pDoc = pDomImpl->createDocument(xhtmlns,
!   "html", pDomImpl->createDocumentType("", "html",
!   "xhtml-sample-1.dtd"));
DocType &= DomImpl.CreateDocumentType(s.c(''),s.c('html'),|
   s.c('xhtml-sample-1.dtd'))
pDoc &= DomImpl.CreateDocument(xhtmlns,s.c('html'),DocType)
```

Now that you have the document, you can start creating its XML elements. This code gets the root node, and adds a comment which will appear before the root node in the document:

```
!CSL::XML::Element* pDocElem = pDoc->getDocumentElement();
pDocElem &= pDoc.GetDocumentElement()

!CSL::XML::Comment* pCommentElem =
!   pDoc->createComment("CenterPoint/XML Sample");
pCommentElem &= pDoc.CreateComment(s.c('ClarionMag DOM example'))

!pDoc->insertBefore(pCommentElem, pDocElem)->release();

pDoc.InsertBefore(pCommentElem,pDocElem)
pCommentElem.Release()
```

Note that when the comment element is no longer needed you call its `Release` method.

The following code adds the HTML `<head>` and `<title>` tags, along with the `<title>` text. Append the text to the `<title>` tag, and append the `<title>` tag to

the <head> tag:

```
!CSL::XML::Element* pHeadElem =
!   pDoc->createElementNS(xhtmlns, "head");
pHeadElem &= pDoc.CreateElementNS(xhtmlns,s.c('head'))

!CSL::XML::Element* pTitleElem =
!   pDoc->createElementNS(xhtmlns, "title");
pTitleElem &= pDoc.CreateElementNS(xhtmlns,s.c('title'))

!pTitleElem->appendChild(pDoc->createTextNode(
!   "XHTML Sample"))->release();
pText &= pDoc.CreateTextNode(s.c('XHTML Sample'))
pTitleElem.appendChild(pText)
pText.Release()

!pHeadElem->appendChild(pTitleElem)->release();
pHeadElem.AppendChild(pTitleElem)
pTitleElem.Release()

!pDocElem->appendChild(pHeadElem)->release();
pDocElem.AppendChild(pHeadElem)
pHeadElem.Release()
```

Now create the <body> tag:

```
!CSL::XML::Element* pBodyElem = pDoc->createElementNS(xhtmlns, "body");
pBodyElem &= pDoc.CreateElementNS(xhtmlns,s.c('body'))
```

Create a paragraph:

```
!CSL::XML::Element* pParaElem = pDoc->createElementNS(xhtmlns, "p");
pParaElem &= pDoc.CreateElementNS(xhtmlns,s.c('p'))

!pParaElem->setAttribute("align", "center");
pParaElem.SetAttribute(s.c('align'),s.c('center'))
```

Create the paragraph text and append to the paragraph tag:

```
!pParaElem->appendChild(pDoc->createTextNode(
!  "Created with CenterPoint/XML."))->release();
pText &= pDoc.CreateTextNode(s.c('Created with CenterPoint/XML.'))
pParaElem.AppendChild(pText)
pText.Release()
```

Append the paragraph to the <body> tag:

```
!pBodyElem->appendChild(pParaElem)->release();
pBodyElem.AppendChild(pParaElem)
pParaElem.Release()

!pDocElem->appendChild(pBodyElem)->release();
pDocElem.AppendChild(pBodyElem)
pBodyElem.Release()
```

Create a DOM writer and write the file to disk:

```
!CSL::XML::DOMWriter domWriter;
Writer &= CreateDOMWriter()

!// enable pretty-printing, no XML declaration
!domWriter.setFormat(CSL::XML::DOMWriter::REFORMATTED |
   CSL::XML::DOMWriter::CANONICAL);
Writer.setFormat(format:reformatted + format:canonical)

!domWriter.writeNode("test.xhtml", pDoc);
if Writer.writeNode(s.c('domtest.xml'),pDoc).

!pDoc->release();
pDoc.Release()
```

Finally, clean up the implementation:

```
DestroyDomImplementation(DomImpl)
```

And you thought the XMLGenerator code was verbose! Welcome to the world of DOM. Of course, if you have a lot of repetitive actions you could wrap this code in another class and save yourself a lot of coding time.

The RSS example

In a previous article I showed how to create an RSS feed in Clarion, using the all-Clarion `XMLGenerator` class. Here's the DOM version of that code:

```
DOMRss               PROCEDURE                                  !

DOMImpl      &DomImplementation
docType      &DocumentType
pDoc         &Document
pText        &Text
pRootElem    &Element
pElement     &Element
pCommentElem &Comment
pChannelElem &Element
pItemElement &Element
writer       &DomWriter
pCData       &CDATASection
encoding     cstring('UTF-8')
s            cciCStringFactory
  CODE
  DomImpl &= CreateDomImplementation()
  DocType &= null
  pDoc &= DomImpl.CreateDocument(s.c(''),s.c('rss'),DocType)

  pRootElem &= pDoc.GetDocumentElement()
  pRootElem.SetAttribute(s.c('version'),s.c('0.91'))
  pCommentElem &= pDoc.CreateComment(s.c('ClarionMag DOM RSS example'))
  pDoc.InsertBefore(pCommentElem,pRootElem)
  pCommentElem.Release()

  pChannelElem &= pDoc.CreateElement(s.c('channel'))

  pElement &= pDoc.CreateElement(s.c('title'))
```

```
  pText &= pDoc.CreateTextNode(s.c('Clarion News'))
  pElement.appendChild(pText)
  pText.Release()
  pChannelElem.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('description'))
  pText &= pDoc.CreateTextNode(s.c('News, product announcements, |
    & 'and other items of interest to Clarion developers'))
  pElement.appendChild(pText)
  pText.Release()
  pChannelElem.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('language'))
  pText &= pDoc.CreateTextNode(s.c('en-us'))
  pElement.appendChild(pText)
  pText.Release()
  pChannelElem.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('link'))
  pText &= pDoc.CreateTextNode(s.c('http://www.clarionmag.com'))
  pElement.appendChild(pText)
  pText.Release()
  pChannelElem.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('copyright'))
  pText &= pDoc.CreateTextNode(s.c('Copyright 1999-2003 by CoveComm Inc.'))
  pElement.appendChild(pText)
  pText.Release()
  pChannelElem.AppendChild(pElement)
  pElement.Release

  ! Add first <item>

  pItemElement &= pDoc.CreateElement(s.c('item'))

  pElement &= pDoc.CreateElement(s.c('title'))
  pText &= pDoc.CreateTextNode(s.c('RPM Email Survey'))
  pElement.appendChild(pText)
  pText.Release()
  pItemElement.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('link'))
  pText &= pDoc.CreateTextNode(s.c('http://www.cwaddons.com/email/'))
  pElement.appendChild(pText)
  pText.Release()
  pItemElement.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('description'))
  pText &= pDoc.CreateTextNode(s.c('Lee White is looking for ' |
    & 'feedback, from current and prospective RPM users, ' |
    & 'about email support in RPM.'))
  pElement.appendChild(pText)
  pText.Release()
  pItemElement.AppendChild(pElement)
  pElement.Release
```

```
  pChannelElem.AppendChild(pItemElement)
  pItemElement.Release()

  ! Add second <item>

  pItemElement &= pDoc.CreateElement(s.c('item'))
  pElement &= pDoc.CreateElement(s.c('title'))
  pText &= pDoc.CreateTextNode(s.c('True Edit-In-Place Template'))
  pElement.appendChild(pText)
  pText.Release()
  pItemElement.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('link'))
  pText &= pDoc.CreateTextNode(s.c('http://www.audkus.dk'))
  pElement.appendChild(pText)
  pText.Release()
  pItemElement.AppendChild(pElement)
  pElement.Release

  pElement &= pDoc.CreateElement(s.c('description'))
  pText &= pDoc.CreateTextNode(s.c('This new  EIP Template ' |
    & 'adds full template support for the Clarion ' |
    & 'edit-in-place list box. ABC templates only; ' |
    & 'includes source and future updates.'))
  pElement.appendChild(pText)
  pText.Release()

  pCData &= pDoc.CreateCDATASection(s.c('This is some text in a CDATA section'))
  pElement.AppendChild(pCData)
  PCData.Release()

  pItemElement.AppendChild(pElement)
  pElement.Release

  pChannelElem.AppendChild(pItemElement)
  pItemElement.Release()


  pRootElem.AppendChild(pChannelElem)
  pChannelElem.Release()


  Writer &= CreateDOMWriter()
  Writer.setFormat(format:reformatted)
  if Writer.writeNode(s.c('domrss.xml'),pDoc).
  pDoc.Release()

  DestroyDomImplementation(DomImpl)
```

This code creates the following RSS file:

```
<?xml version="1.0"?>
<!--ClarionMag DOM RSS example-->
<rss version="0.91">
  <channel>
    <title>Clarion News</title>
    <description>News, product announcements, and other
      items of interest to Clarion developers</description>
    <language>en-us</language>
```

```
    <link>http://www.clarionmag.com</link>
    <copyright>Copyright 1999-2003 by CoveComm Inc.</copyright>
    <item>
      <title>RPM Email Survey</title>
      <link>http://www.cwaddons.com/email/</link>
      <description>Lee White is looking for feedback, from
      current and prospective RPM users, about email support in
      RPM.</description>
    </item>
    <item>
      <title>True Edit-In-Place Template</title>
      <link>http://www.audkus.dk</link>
      <description>This new  EIP Template adds full
      template support for the Clarion edit-in-place list box.
      ABC templates only; includes source and future updates.
      <![CDATA[This is some text in a CDATA section]]></description>
    </item>
  </channel>
</rss>
```

There are a couple of things to note about this version as compared to the `XMLGenerator` version. First, I was able to add a version attribute to the root tag using this code:

```
pRootElem &= pDoc.GetDocumentElement()
pRootElem.SetAttribute(s.c('version'),s.c('0.91'))
```

That meant I didn't have to process the file a second time to make corrections, as I did with `XMLGenerator`. Second, the file is nicely formatted thanks to the `DOMWriter` object. Pretty printing may not make much difference to any RSS reader software I use but I like it better that way. And finally, just to see that it works, I've added a `CDATA` section to one of the descriptions, using this code:

```
pCData &= pDoc.CreateCDATASection(|
  s.c('This is some text in a CDATA section'))
pElement.AppendChild(pCData)
```

There is, as of this writing, no way to create `CDATA` sections using `XMLGenerator`, although apparently this capability will be added.

Clearly there's a lot of grunt work involved in creating XML files using DOM. In many cases you will want to create a wrapper class for your specific requirements; for instance, an RSS class could simply have one method to create all of the channel description tags, and another method to be called for each item to be added to the RSS file.

Summary

Although I haven't had a chance to explore reading XML files with the C6 CenterPoint XML DOM (or SAX) parser, I didn't run into any difficulties (aside from the `CSTRING` literals issue) writing XML using the DOM wrapper, and that suggests that the Clarion

layer is in pretty good shape. Some documentation is available from the CenterPoint web site, but DOM, like SAX, is a standard, so you may be best off just going out and getting a good book on the subject.

Download the source

Download the cciCStringFactory class (required)

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of Developing Clarion for Windows Applications, published by SAMS (1995). His most recent book is JSP, Servlets, and MySQL, published by HungryMinds Inc. (2001).

Reader Comments

Add a comment