Clarion MAGAZINE

Search
Go

Home

COL Archives

Subscribe
New Subs
Renewals

Info
Log In
FAQ
Privacy Policy
Contact Us

Downloads
PDFs
Freebies
Open Source

Site Index

Call for Articles

A Template for Preserving Procedure Variables

Clarion 5 introduced nice new feature in the standard ABC templates, and that was the ability to easily preserve the contents of global variables in the application's INI file. Tom Hebenstreit shows how to apply this idea to procedure data.

Posted Wednesday, January 31, 2001

Windows-Style List Box Sorting

Clarion's standard way of sorting a list box is to use a sheet with tabs for each key. You select a tab to apply the desired sort order. In other Windows programs this same functionality you sort by clicking on the list box header, so why not do the same in Clarion?

Posted Wednesday, January 31, 2001

Tell Us Where To Go!

We want to know what kinds of articles you'd like to see in Clarion Magazine. Do you want more introductory-level material? More in-depth, highly technical articles? More news and interviews? What topics would you like Clarion Magazine to address?

Posted Wednesday, January 31, 2001

Data Is Executed Policy - A Case Study

This Whitemarsh paper presents a Clarion-centric approach, illustrated through a case-study, to requirements analysis and design through production system implementation. This material can be given to clients in support of a systems development proposal.

Posted Thursday, January 25, 2001

Look Ma, No Keys!

With ABC you can sort a browse on fields other than those defined in your keys, with reasonably good performance in most cases. Unfortunately you also lose locators

HTML Designer Updated

RSBackUp Version 1.0 Released

ZIPFlash 2.0 Released

HTML Help Designer Now Supports Clarion 2003

Gitano Software Office Closed January 22-28

Free PDF Organizer From Gitano Software

US Postnet Barcodes for Clarion 5.5

File Explorer Special Ends In Two Weeks

Imaging Templates Upgrade Available

Solace WordSpell Templates Beta 3 Released

SocketTools 3.5 Service Pack 3 Released

CapeSoft File Explorer 1.0 Goes Gold

and scrolling thumb support. Jim Kane goes digging in ABC and discovers it's not that difficult to add back the locator and thumb.

Posted Tuesday, January 16, 2001

Sending Clarion Reports as Email Attachments (Part 2)

The email capability in version 5.5 is a nice addition to the Clarion toolset. What is still missing however, is the ability to easily send a report as an email attachment. In this article David Potter demonstrates one possible solution to this problem. Part 2 of 2.

Posted Tuesday, January 16, 2001

My Name Is Tom, And I'll Be Your Server

It's been a long Christmas break for ClarionMag readers, but the first issue of 2001 is now available on the new Tomcat-powered ClarionMag server.

Posted Tuesday, January 09, 2001

Sending Clarion Reports as Email Attachments (Part 1)

The email capability in version 5.5 is a nice addition to the Clarion toolset. What is still missing however, is the ability to easily send a report as an email attachment. In this article David Potter demonstrates one possible solution to this problem. Part 1 of 2.

Posted Tuesday, January 09, 2001

Implementing SELECT DISTINCT in a TPS Database

In SQL when you want a report showing only single instances of a particular data element you can use the SELECT DISTINCT statement. But what do you do when you want this same capability in a TPS database? Jon Waterhouse presents a template that does the job.

Posted Tuesday, January 09, 2001

January 2001 News

Clarion news for January, 2001

Posted Monday, January 01, 2001

Automated Fax Engine for 5.5 Gold Release Available

Freeware: The Sterling Data Calculator

Templates Integrate MSWord SpellCheck

Search
Go

Home

COL Archives

Subscribe
New Subs
Renewals

Info
Log In
FAQ
Privacy Policy
Contact Us

Downloads
PDFs
Freebies
Open Source

Site Index

Call for Articles

A Template for Preserving Procedure Variables

by Tom Hebenstreit

Published 2001-01-31

Clarion 5 introduced nice new feature in the standard ABC templates, and that was the ability to easily preserve the contents of global variables in the application's INI file. You clicked on the 'Preserve' button in the Global options, chose the variables from the list and voila, their contents were saved when you exited the application and restored when you run it.

I found this feature very handy, but I had the feeling that the concept could be extended even farther. Specifically, I wanted the same functionality at the procedure level as well.

I often have local procedure variables that I want to maintain between calls to a procedure (and/or program execution), and manually writing embed code to do the saving and restoring is an exercise in grunt work. For example, in many wizards I want to save the user's choices so that they didn't have to re-specify the same options over and over. I also want to preserve certain entries on forms as user defaults whenever a new form was inserted.

Hmmmm. Tedious coding...grunt work...sounds like a job for...A TEMPLATE!

And that's just what I'll be discussing here – a simple procedure extension template that does the following:

- Allows you to save and restore any variable or field by selecting it from a list
- Allows you to specify the INI section to save the values in
- Allows you to specify virtually any default value for each variable
- Has a special option for forms to only restore when a record is being inserted

Note: Although I'll be using ABC code in the template, I've included some comments at the end to help legacy template users convert the template for use with that chain as well. Additionally, all of the information on the template language itself applies equally to either template chain.

First things first

The first step in creating a template like this is to first get the concept working manually using embed code. That way you are only dealing with one task at a time – either writing and testing Clarion code or writing and testing a template to generate that code.

As you would expect, the code to restore values from the INI file needed to go into my procedure's `WindowManager.Init` method, and the save code to put them back into the INI needed to be placed in the `WindowManager.Kill` method.

First off, you need to declare the variables to save and restore using the 'Data' button on your procedure properties. Then, in the `Init` method you place code like this to restore a saved value (if any):

```
WizSkipDetails = True ! Default variable to True
IniMgr.Fetch('Preference','WizSkipDetails',WizSkipDetails)
```

Note this version of the `IniMgr.Fetch()` method allows you to provide a default value. Unlike using the Clarion language GETINI() function directly, where you can pass variable or a constant, the ABC IniMgr requires that you pass a variable, and it uses the value of that variable as the default if it doesn't find the INI entry.

To save the current value, you do something like this the Kill method:

```
If Self.Response = RequestCompleted
  IniMgr.Update('Preference','WizSkipDetails',WizSkipDetails)
End
```

Notice how saving the value is wrapped inside a test to see if the procedure completed normally. That way you avoid replacing known good values with, for example, partially filled or blank variables in cases where the user got part way through the wizard and then cancelled.

After entering the code manually, you can use the Source button to see exactly where to tell the template to generate the code (i.e., where to place it and at what priority level, etc.).

Examining the template

First, let's take a look at how the template looks when used within the Clarion Application Generator.

Once you've registered the template, you can use it with any procedure by inserting it under your procedure properties 'Extensions' button. Figure 1 below shows the template with several variables already added to the list, along with their default values.

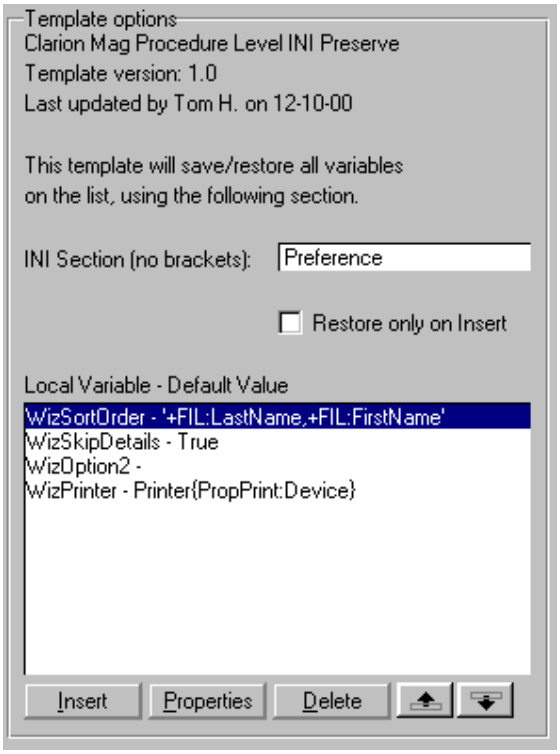

Figure 1. The template prompts and list of variables

Pressing Insert or Properties buttons to bring up a small window where you can select the variables to preserve as well as provide an optional default value.

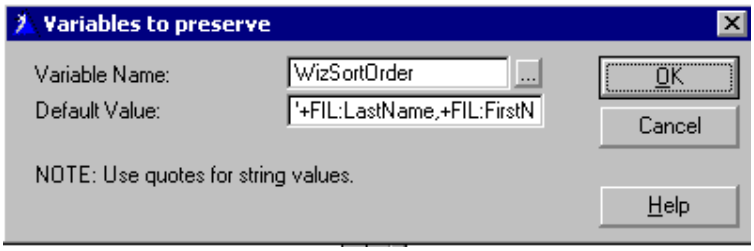

Figure 2. Specifying a variable to preserve

Now let's go under the hood and see what makes this template tick.

Following this is the entire template broken down into three main sections, with commentary on each section. Note that I will be approaching this at a rather high level rather than picking apart each template statement in exhaustive detail. Use the on-line help to delve more deeply into the options that are available for any given statement.

The header sections identify the templates and provide the names you see when you select a template from the list of templates.

```
#!************************************************
#TEMPLATE(ClarionMag,'Clarion Magazine Templates')↵
  ,FAMILY('ABC')
#!************************************************
#!
#EXTENSION(ProcIniPreserve,'Procedure level ↵
  INI Preserve v1.0'),WINDOW
```

The #TEMPLATE statement and TPL extension on the file mean that this is a standalone template, and can be registered all by itself. I usually prefer to have another TPL file where I group small templates like this, and just set that file to #INCLUDE the extension. That way there is only one file to register rather than handfuls of files, and all I need to do is add an #INCLUDE line to the TPL file in order to automatically register a new little template.

For example, I have a TPL file that looks like this:

```
#!************************************************
#TEMPLATE(MiscTemplates,'Misc Template Collection')↵
  ,FAMILY('ABC')
#!************************************************
#INCLUDE('\Common\MiscRpt.TPW')
#INCLUDE('\Common\MiscFrm.TPW')
#INCLUDE('\Common\MiscIni.TPW')
#! more #INCLUDE statements. . .
#!************************************************
```

This TPL file provides the #TEMPLATE line, so all I need are the associated TPW files that contain the actual template code. Remember, TPL files can be self-contained (like this small example) or may have dozens of supporting TPW files that contain other templates or support code (see the ABCHAIN.TPL in your Clarion \Template folder for a mondo example). You don't register TPW files, you just register the TPL that #INCLUDEs the TPWs.

Next in the template is the part that creates all of the interaction you saw in Figures 1 and 2:

```
#!====== Template Prompts =======
#BOXED('Template Options')
  #DISPLAY('Clarion Mag Procedure Level INI Preserve')
  #DISPLAY('Template version: 1.0')
  #DISPLAY('Last updated by Tom H. on 12-10-00')
  #DISPLAY(' ')
  #DISPLAY('This template will save/restore all variables')
  #DISPLAY('on the list, using the following section.')
  #DISPLAY(' ')
  #PROMPT('INI Section (no brackets):',@S25), ↵
    %PreserveSection,DEFAULT('Preference')
  #DISPLAY(' ')
  #PROMPT('Restore only on Insert',CHECK),%PreserveOnInsert
  #DISPLAY(' ')
  #DISPLAY('Local Variable - Default Value')
  #BUTTON ('Variables to preserve'), MULTI(%PreserveVars, ↵
    %PreserveVar & ' - ' & %PreserveDefault), INLINE
    #PROMPT('Variable Name:',FIELD),%PreserveVar
    #PROMPT('Default Value:',@S45),%PreserveDefault
    #DISPLAY(' ')
    #DISPLAY('NOTE: Use quotes for string values.')
  #ENDBUTTON
#ENDBOXED
```

Take another look at Figure 1 and you'll see that everything that appears on that window is generated from this little block of template

code.

And what do we have here? Every #DISPLAY statement does just that, display something on the template options window. The #PROMPT statements get input such as the INI section to save to, the variables to save and default values.

The #BUTTON statement is especially powerful in that it automatically creates and manages the list where we specify the variables, complete with the Insert, Properties, and Delete buttons and the arrows for changing the sequence. The #PROMPT statements between the #BUTTON and #ENDBUTTON generate the little popup form you saw in Figure 2.

As you can see, you can actually generate quite a bit of user interaction with just a few lines of template code. In fact, if you take away the #DISPLAY statements used for instructions, etc., the template still works with only 6 lines of code (four #PROMPT statements and #BUTTON/#ENDBUTTON statements).

Now for the meat of the matter – generating the code. Take a look at the first two blocks of template code together, since they do exactly the same thing (only in different places):

```
#!-------------------------------------------------
#AT( %WindowManagerMethodCodeSection, 'Init', ↵
  '(),BYTE'), PRIORITY(2001),WHERE(~%PreserveOnInsert)
#FOR(%PreserveVars)
#IF( %PreserveDefault )
%PreserveVar = %PreserveDefault
IniMgr.Fetch('%PreserveSection','%PreserveVar',↵
  %PreserveVar)
#ELSE
%PreserveVar = IniMgr.TryFetch('%PreserveSection','↵
  %PreserveVar')
#ENDIF
#ENDFOR
#ENDAT

#!-------------------------------------------------
#AT( %WindowManagerMethodCodeSection, 'PrimeFields'), ↵
  PRIORITY(6001),WHERE(%PreserveOnInsert)
#FOR(%PreserveVars)
#IF( %PreserveDefault )
%PreserveVar = %PreserveDefault
IniMgr.Fetch('%PreserveSection','%PreserveVar',↵
  %PreserveVar)
#ELSE
%PreserveVar = IniMgr.TryFetch('%PreserveSection',↵
  '%PreserveVar')
#ENDIF
#ENDFOR
#ENDAT
```

The #AT statement determines the method and/or embed point where the code is going to be placed, and the PRIORITY parameter indicates where it goes within that method or embed, e.g., before the window is opened, before files are opened, etc.

In this case, the first block goes into the WindowManager.Init() method (where it will always be executed), while the second block goes into the WindowManager.PrimeFields() method (which is only executed when you are inserting a new record.)

The WHERE parameter for #AT provides a way to only generate code in a particular instance. In this case, the first block of code (for 'Init') is generated only if you didn't check the 'Restore only on Insert' box. By the same token, the code for 'PrimeFields' is only generated if you *did* check the box. Based on the value of the template %PreserveOnInsert variable, you'll always get one or the other – never both.

In essence, you have created a big IF/ELSE type of test. If you're using this code in, for example, a wizard, you would want the code to execute every time the procedure is called. If you're using the template on a form, you would check the 'only on Insert' box. Why? So that you don't overwrite existing values with default ones if the user is *updating* a record rather than adding one.

By the way, I strongly suggest you read up on #AT, as it is arguably one of the most important template statements used by extension templates.

Moving on, if you ignore the #AT lines you'll notice that the two blocks that generate Clarion code are identical. Each uses a #FOR statement to loop through the list of variables you specified on the template prompts. Depending on whether a default value was provided, this code uses either the IniMgr.TryFetch() or IniMgr.Fetch() method to retrieve a value.

You've declared the template, gathered the prompt info and restored the contents of any preserved variables. All that is left is to add the block of code that saves the variable contents (but only when the user didn't cancel out of the procedure).

```
#!-----------------------------------------
#AT( %WindowManagerMethodCodeSection, 'Kill',↵
  '(),BYTE'),PRIORITY(2001)
If Self.Response = RequestCompleted
#FOR(%PreserveVars)
  IniMgr.Update('%PreserveSection',↵
    '%PreserveVar',%PreserveVar)
#ENDFOR
End
#ENDAT
```

By now this code should be looking familiar to you. You have an #AT statement to place the code in the WindowManager.Kill method, and a #FOR loop to run through the list and generate the IniMgr.Update() line that saves the current contents of each variable.

Using the example variables and defaults from Figure 1, this is what's generated into the WindowManager.Init() method of that procedure:

```
WizSortOrder = '+FIL:LastName,+FIL:FirstName'
IniMgr.Fetch('Preference','WizSortOrder',WizSortOrder)
```

```
WizSkipDetails = True
IniMgr.Fetch('Preference','WizSkipDetails',WizSkipDetails)
WizOption2 = IniMgr.TryFetch('Preference','WizOption2')
WizPrinter = Printer{PropPrint:Device}
IniMgr.Fetch('Preference','WizPrinter',WizPrinter)
```

Notice how the variable WizOption2 uses the alternate method of fetching the INI value since there's no default value. You can use any valid Clarion statement to assign the default, including equates and properties.

When it comes to saving the variables, here is what's generated in the `WindowManager.Kill()` method:

```
If Self.Response = RequestCompleted
  IniMgr.Update('Preference','WizSortOrder',WizSortOrder)
  IniMgr.Update('Preference','WizSkipDetails',WizSkipDetails)
  IniMgr.Update('Preference','WizOption2',WizOption2)
  IniMgr.Update('Preference','WizPrinter',WizPrinter)
End
```

And that's all there is to it.

For the Legacy crowd

Although I've been using ABC code in the template so far, the basic concepts apply equally well to legacy template users. In fact, if you use a legacy version of the template you can add the same type of functionality as the ABC built-in 'Preserve' option. Just attach the template to your frame and specify your global variables.

The first change to make is to change the `FAMILY(ABC)` option on the `#TEMPLATE` statement to:

```
#TEMPLATE(ClarionMag,'Clarion Magazine Templates')↵
  ,FAMILY('CW20')
```

The `FAMILY()` option determines whether a particular template chain appears on your list of templates, based upon whether you are using the ABC or Legacy templates.

Next, you change `AT#` statements so that they pointed to legacy embeds rather than ABC methods.

Finally, you simply replace the `IniMgr` calls with stock `GETINI()` and `PUTINI()` statements. For example, to restore a variable you use:

```
%PreserveVar = %PreserveDefault
%PreserveVar = GetINI('%PreserveSection',↵
  '%PreserveVar',%PreserveVar, '%INIFileName')
```

And for saving a value:

```
PutINI('%PreserveSection','%PreserveVar',%PreserveVar,↵
  '%INIFileName')
```

Possible Improvements

You know, tinkering with small templates is a great way to gain experience with the template language, and a simple procedure extension like this is a good entry into the often mystical but extremely powerful world of templates.

As usual, there are probably many little improvements that can be made to this template. You might let the developer set the priority level for the embed points. This would let you move the point where the code is inserted so that, for example, in one procedure it would be generated after the window has been opened and before files are opened in another (the template currently generates the code fairly early within the Init and Kill methods.) I'm sure you can think up a few enhancements of your own, too.

Jump in and enjoy!

Download the template

A longtime Clarion user, Tom Hebenstreit is an admitted tool junkie who refuses to go straight and code without his arsenal of third party products. During those rare moments when he isn't either using or writing about Clarion, he indulges his twin passions for blues and beer by performing around Southern California in a variety of totally-obscure-but-famous-any-day-now rock and blues bands.

Reader Comments

Add a comment

Windows-Style List Box Sorting

by Steffen Rasmussen

Published 2001-01-31

Clarion's standard way of sorting a list box is to use a sheet with tabs for each key. You select a tab to apply the desired sort order. In other Windows programs this same functionality you sort by clicking on the list box header, so why not do the same in Clarion? Of course there are third party products that accomplish this and a lot more. But I just need to sort the list box and do nothing else, except show a small icon that tells which column is sorted and in which direction.

Actually this is easier said than done. After working on and off on this small project, experimenting and searching the internet for different solutions, I could not find a satisfactory approach that didn't require some kind of compromise to the existing functionality of the list box.

Since I didn't want to lose any functionality I decided to program this behavior into the existing list box, and that's the subject of this article

Clicking on the header

If you go digging through the Language Reference you'll find a small chapter on the list box Mouse Click Properties. This chapter describes some `PROPLIST` runtime properties which return the mouse position within the List box. `PROPLIST:MouseDownRow` tells you which row is selected by the user and `PROPLIST:MouseDownField` tells you which field.

The next step is to implement the code to add this functionality. Luckily the Language Reference includes a partial example on how to use these commands.

First you have to specify a key to trigger an event within the list box. The usual choice is the left mouse click on the header. To set the Alert key right click the list box and select Alert, as shown in Figure 1.

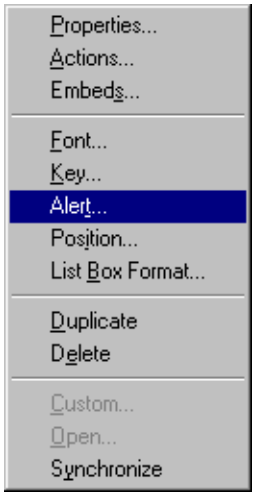

Figure 1. Choose Alert from the right-click context menu

In the Alerts Keys window select Add, as shown in Figure 2.

Figure 2. The Alert Keys window

In the option box Mouse select the radio button Left Button.

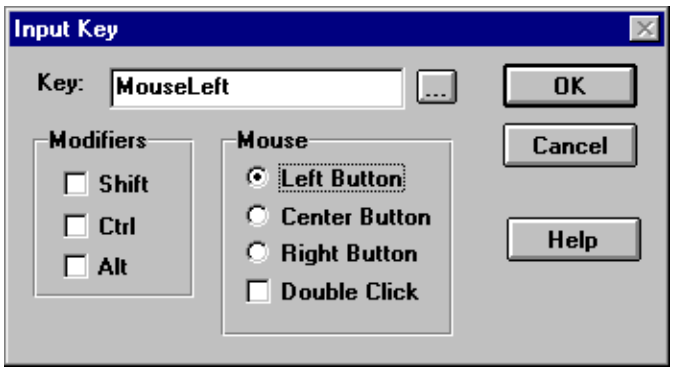

Figure 3. Choosing the input key

To trap the event that is sent to the program when the user clicks the left mouse button you have to check for it in the list box's Control Events `?Browse:6.AlertKey` embed point:

```
IF KEYCODE() = MouseLeft
  IF ?Browse:6{PROPLIST:MouseDownRow} = 0
```

```
    LOC:SortKey = ?Browse:6{PROPLIST:MouseDownField}
  ELSE
    BRW6.TakeNewSelection()
  END
END
```

When you've caught the Mouse Left key event you have to find out which row the user selected in the list box. Use `PROPLIST:MouseDownRow` to determine if the left mouse click was in the list box header. A value of 0 confirms a header selection; you will have to pass any other value on to the `BrowseClass` with the `TakeNewSelection` method, because the mouse left click selects the records in the list box.

When the user clicks the left mouse button in the header you will have to find out which column the mouse is over. Here you use the `PROPLIST:MouseDownField` property, which returns the column number. Assign this value to a local variable called `LOC:SortKey`, a `SHORT`. You will use this variable to set the Conditional Browse Behavior.

In the list box right use the right mouse button to bring up the context menu, select Actions and then the Conditional Behavior tab. Press the Insert button.

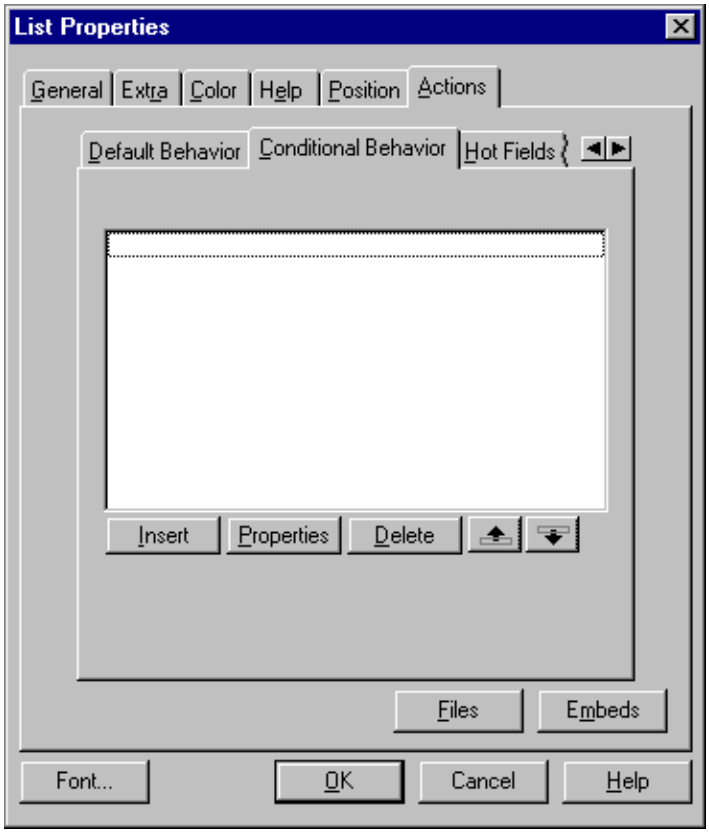

Figure 4. Conditional Browse Behavior

As I previously mentioned, Clarion uses a sheet with tabs to select the sort order. The conditional behavior for each tab looks something like:

```
CHOICE (?Sheet) = tab number
```

but in this case you will use the column number you saved in the `LOC:SortKey` variable:

```
LOC:SortKey = tab number
```

Use the same key as when using the sheet.

If you try to compile and run the application it won't behave as expected. Clicking on the header sorts the rows but you can't select a record by clicking on the row. Why is that? When you set the Alert key for the mouse left button you actually override any other code that uses the mouse left button, and when this code is implemented and executed the code processing of the mouse left key will stop. In maintain normal mouse left button processing after the Alert Key has been triggered you have to implement the following code in the Control Events `?Browse:6.PreAlertKey` embed:

```
IF KEYCODE() = MouseLeft
  CYCLE
END
```

Now you can sort the columns by pressing on the list box header without affecting any other functionality of the list box.

Depressing the header

When you sort the List box by clicking on the header, the only sort indicator is that the list is resorted, and this isn't always noticeable. In order to inform the user that an action actually took place you could make the header look like a flat button, which gets depressed by the user. This is also a good indication to the user whether the column can be sorted or not.

You've probably noticed that if you're using the toolbar to control the Insert, Change and Delete buttons, these buttons are usually placed under the list box and hidden. If you forget to hide these buttons they will jump on top of the list box when they receive focus, which isn't desirable. But you can use this behavior to create the flat button look on the header.

First create a region and place it under the list box. The size of the region is unimportant at this stage. Set the region mode to hide. Select the extra tab, and set the bevel so that the region looks like a depressed button. You can do this with an outer and an inner bevel, each with negative value of one.

You also need to know when the mouse is lifted from the region so that you can refresh the list box. If you look in the embedded source the region has only two embed points: Accepted and All Events. This is because the IMM attribute isn't set for the region control. Go back to the regions extra tab and in the options group check the immediate box. Now you will find several embed points for the region control, which is going to be used later.

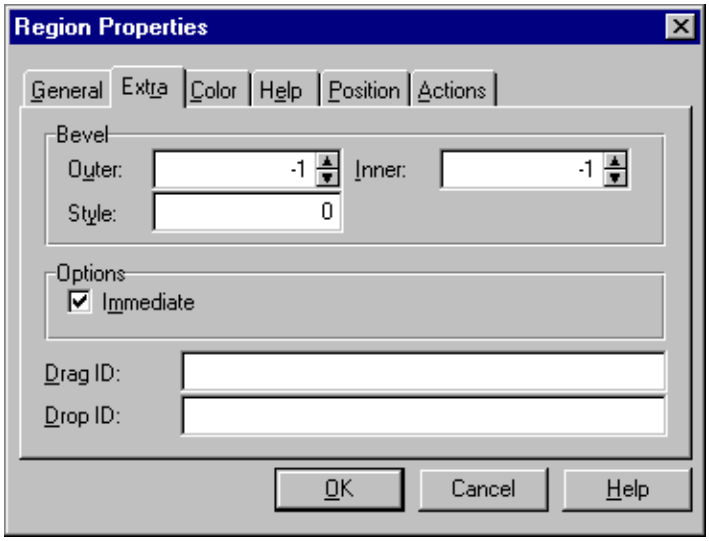

Figure 5. Region Properties Extra tab

In the Control Events `?Browse:6.AlertKey` embed point create the following code:

```
IF  KEYCODE() = MouseLeft
  IF ?Browse:6{PROPLIST:MouseDownRow} = 0
    LOC:SortKey = ?Browse:6{PROPLIST:MouseDownField}
    ?Browse:6{PROP:Edit,LOC:SortKey} = ?Region1
    ?Region1{PROP:YPos} = 0
  ELSE
    ?Browse:6{PROP:Edit,LOC:SortKey} = 0
    BRW6.TakeNewSelection()
  END
END
```

You assign the Region `?Browse:6{PROP:Edit,LOC:SortKey}` as if it were an edit-in-place field. In this way the Region is resized to the same size as the existing column width and line height as well as the same coordinates as the marked record in the list box. `?Region1{PROP:YPos} = 0` changes the Regions position to be the same as the column header.

If the column header has not been selected the Region is hidden :

```
(?Browse:6{PROP:Edit,LOC:SortKey} = 0)
```

Sorting the list box

To be able to sort one column in ascending and descending order the program has to know what the previous sort was in order to determine the sort direction. Since the `LOC:SortKey` already contains the value for the column number you can use a negative number for descending order. For this purpose create a local variable called `LOC:PreSortKey`, `SHORT`. This way you make as few alterations as possible to the existing program structure. Now it is just a matter of determining if the `LOC:SortKey` is equal to the `LOC:PreSortKey` and if it is, shift the value from positive to negative or vice versa.

Modify the code in the Control Events ?Browse:6.AlertKey embed point so it looks like this:

```
IF KEYCODE() = MouseLeft
  IF ?Browse:6{PROPLIST:MouseDownRow} = 0
    LOC:SortKey = ?Browse:6{PROPLIST:MouseDownField}
    ?Browse:6{PROP:Edit,LOC:SortKey} = ?Region1
    ?Region1{PROP:YPos} = 0
    IF LOC:SortKey = LOC:PreSortKey
      LOC:SortKey=LOC:SortKey-(LOC:SortKey*2)
    END
    LOC:PreSortKey = LOC:SortKey
  ELSE
      ?Browse:6{PROP:Edit,LOC:SortKey} = 0
      BRW6.TakeNewSelection()
  END
END
```

The next step is to set the descending sort order for the list box in the Conditional Browse Behavior. The Order would be for example LOC:SortKey = -2. The Key to Use has to contain a key, which is sorted in descending order. If you don't have and don't want this type of key, leave the field blank. Instead enter the ascending key value in the entry field: Additional Sort Fields. To make the field sort in a descending order, a minus (-) sign must precede the file prefix (-CUS:CustNumber). Note that sorting with out the use of a key drastically increases the processing time. Therefore you shouldn't us it on a large table unless you've filtered the records into a smaller, manageable volume.

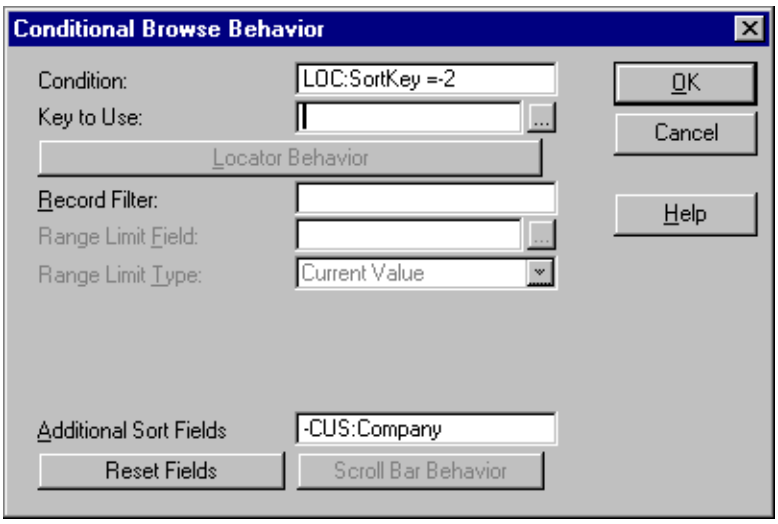

Figure 6. Conditional Browse Behavior

> **Note:** there is a flaw in this code. When the column sorts in ascending and descending order and uses a key and a non key to accomplish this process, the user will not be able to see the header button be depressed, when a field name is used to sort the list box. This is because the change from key to non-key triggers an immediate list update, which I haven't been able to delay.

Showing the sort direction

Now you can sort a column in either ascending or descending order you may want to show the user which direction the column is sorted in. As standard the Clarion List box does not support icons in the list box header, and personally I haven't been able to find a work around, although I am still trying. For the moment Ill just have to use the second best solution.

Since it is only possible to use ASCII characters in the list box header, you just have to find a font that suits that your need. In this particularly case I am using the font MS Sans Serif and the two characters » « to represent ascending and descending order.

The header text can contain three different instances:

1. Header text (no sorting)
2. '» ' & Header text (sort in ascending order)
3. '« ' & Header text (sort in descending order)

By default the header only contains text, so in order to make this work you will have to save, add, change and reset the header.

Implement this in the Control Events `?Browse:6.AlertKey` embed point:

```
IF KEYCODE () = MouseLeft
  IF ?Browse:6{PROPLIST:MouseDownRow} = 0
    LOC:SortKey = ?Browse:6{PROPLIST:MouseDownField}
    ?Browse:6{PROP:Edit,LOC:SortKey} = ?Region1
    ?Region1{PROP:YPos} = 0
    IF LOC:SortKey = LOC:PreSortKey
      LOC:SortKey=LOC:SortKey-(LOC:SortKey*2)
      ?Browse:6{PROPLIST:Header,ABS(LOC:SortKey)} |
        = '« ' & LOC:PreHeader
    ELSE
      ?Browse:6{PROPLIST:Header,ABS(LOC:PreSortKey)} |
        = LOC:PreHeader
      LOC:PreHeader = |
        ?Browse:6{PROPLIST:Header,ABS(LOC:SortKey)}
        ?Browse:6{PROPLIST:Header,ABS(LOC:SortKey)} |
        = '» ' & LOC:PreHeader
    END
      LOC:PreSortKey = LOC:SortKey
  ELSE
    ?Browse:6{PROP:Edit,LOC:SortKey} = 0
    BRW6.TakeNewSelection()
  END
END
```

If the user selects the same header, then the sort direction is changed and so is the header text instance. If the user selects a different header, the previous selected header has to be reset with its original text with out the leading sort direction symbol (» «). The next step is to assign the new header string to the `LOC:PreHeader` variable. After this you add the new direction symbol to the header.

Because of this change of the header text you have to initialize the `LOC:PreHeader` when the window is opened.

Implement the following in the Window Events `OpenWindow` embed point:

```
LOC:PreHeader = ?Browse:6{PROPLIST:Header,ABS(LOC:SortKey)}
```

As you can see the code contains the variable `LOC:SortKey`. You could substituted a constant value of 1, but for readability I prefer to set the `LOC:SortKey` initial value to 1 in the local data section of the procedure.

That's it, except for a few flaws. I have already mentioned the non-key sort problem. Another is that this code assumes that all columns can be sorted in both ascending and descending direction. So clicking in a column, which does not contain any conditional browse behavior, will also show the button selection as well as the sort direction, which in this case does not exist.

Steffen S. Rasmussen has graduated in Computer Science from Copenhagen Business College. Since then he has worked as a programmer, system technician and network administrator, and is currently IT manager. Clarion is a quite a new language to Steffen since his only been working with it since January 2000. But what better way to learn it than by trying to teach others! Steffen has also set up a web site to collect as many examples of different user interfaces as possible to inspire Clarion developers.

Reader Comments

Add a comment

Sending Clarion Reports as Email Attachments (Part 2)

by David Potter

Published 2001-01-16

Part 2 of 2

In Part 1 of this two-part series I explained how to create a report that can be emailed to a user. In this article I'll wrap everything up by actually emailing the report. The email procedure included in the example programs supplied with the Gold release is very close to what I need, so I'll do what any self respecting programmer would do: I'll steal it and make minor changes to better suit my needs. These changes are as follows:

1. Add the global email extension to the program containing the report and fill out the necessary fields.
2. Import the `SendProc` procedure from `MessageTest` application located in the `..\examples\messaging\all` directory of the C55 gold installation.
3. Once imported, add the prototype `(STRING,STRING)` and the parameters `(pAttachment,pReportName)` to the procedure. This will allow me to pass the name of the metafile file to be attached as well as a report identifier to be used in any headers.
4. Add the local string variable `Loc:Attachment` to the Local Data List in the `SendProc` Procedure. Then enter `!Loc:Attachment` to the attachment list in the filename entry, in the extension under the Attachment tab.
5. Set the local variables in the `ThisWindow.Init` embed:

```
Loc:Attachment = pAttachment
Loc:Priority = 'High'
Loc:Subject = pRptName & ' For ' |
  & FORMAT(TODAY(),@D2)
Loc:MessageText = |
  'Attached you will find the latest ' |
  & LOWER(pRptName) & '|' |
  & 'Please double click on the icon below ' |
  & '| Press Launch.. to view the report'
```

Adding a recipient list

Since the report goes to the same group of people every day, it would be a nice touch to fill the recipient list so the user wouldn't have to do it. In order to do this I have set up three related tables. One listing all of the employees with their email addresses, a second listing all of the reports we use in the company and a third joining the reports with the employees to create a recipient list. By looping through the Employee table I can fill the `GlobalAddressBook` using the `GlobalAddress.AddRecipient` method. By checking the passed `pRptName` parameter I can fill a recipient list, including only the employees linked to that particular report, using the `Address.AddRecipient` method.

The `MessageTest` application has an excellent example showing how to use a process to fill the Global and Local `AddressBooks` in the `BatchEmail` procedure.

Calling the SendProc procedure

Now I just need to add a call to the email procedure from the report. I can do this in the `Previewer.Display` embed right after finishing the loop.

```
Relate:Metafile.Close
SendProc(Glo:MetaFileName,'MyReport')
REMOVE(Metafile)
```

This code calls the `SendProc` procedure, passing the name of the TopSpeed file to be sent as an attachment. The second parameter will be used to get the recipient list for the report. It can be used in the header or body of the email as well. Upon returning from the Email procedure the renamed tps file can be deleted as it is no longer needed.

Finally I will create a global flag, `GLO:SendAsEmail`, that can be wrapped around the code in the `Previewer.Display` embed. By setting the flag to false I can bypass everything and print and view the report normally. All that is necessary is to sandwich the code in an IF statement.

```
IF GLO:SendAsEmail THEN
  Previewer.Display Code ...
END
```

The `GLO:SendAsEmail` variable can also be used to ensure that the `SkipPreview` property for the report has not been set. Including the following code in the `ThisWindow.Init` embed at the highest priority (9500) will ensure that the `Previewer.display` method will always be called:

```
IF Glo:SendAsEmail
  SELF.SkipPreview = False
END
```

Now the report can optionally be sent to the printer, or attached to an email, simply by setting a global flag in the report parameter window.

Configuring the clients

The only thing left to do now is to set up in the individual client machines. When someone receives a report sent as an attachment, all I want them to have to do is double click on the icon. Then the viewer will open the file, preview the report and print it as necessary. As I stated earlier, this requires an association between a file extension and the viewer program. This is easy to do on an individual machine but can be very time consuming on a large network. By studying the registry, I found the entries made when I manually created the association. By using standard Windows API calls, I can create a program to update the registry. Then by including this program in the network login script I was able to get every machine on the network to create the association automatically.

There are many ways to update the registry. The API calls are fairly easy to prototype. There are also many third party templates that can make this easier. I used the `apiRegistryClassGlobal` template from `ABCFreeTemplates`, which saved me from having to look up and prototype the necessary API calls.

The following steps will create the necessary program:

1. Set the default in Application Options to not require a dictionary.
2. Add the ABCFree template under Global Extensions.
3. Create a main procedure consisting of one source procedure with the following four lines of code.

```
IF registry.Fetch(HKEY_LOCAL_MACHINE,|
 'SOFTWARE\Classes\PVW_auto_file','Version')
  Registry.Update(HKEY_LOCAL_MACHINE,|
   'SOFTWARE\Classes\.PVW','','PVW_auto_file')
  Registry.Update(HKEY_LOCAL_MACHINE,|
   'SOFTWARE\Classes\PVW_auto_file',|
   '',' Report Viewer')
  Registry.Update(HKEY_LOCAL_MACHINE,|
   'SOFTWARE\Classes\PVW_auto_file',|
   'Version','1')
  Registry.Update(HKEY_LOCAL_MACHINE,}
   'SOFTWARE\Classes\PVW_auto_file\shell\open\command',|
   '','J:\CLARSYS\WINAPPS\metaview.exe %1')
END
```

I added the version parameter as a check: if the registry has already been updated it won't be again. The particular parameters of this program will vary with each implementation but this will get you started.

You've read the code but have you seen the template

This procedure as described has been in use now for several weeks and everyone has been pleased with it. As stated earlier, the fact that the reports are being sent internally allows us to store all of the metafiles in a central location. However it would be very possible to devise a method of compressing the metafiles and sending them along with the file list. This will have to wait until there is a need to send a report to someone not on our WAN.

It would also be possible to wrap the viewer with the other files and send it as a single exe. This would alleviate having to do any setup on the recipient machine.

The last thing that still needs to be done at our site is to create a cleanup program to periodically go through the metafile directory and delete old files. Currently we are doing this manually every week or so but there is no reason not to develop a program to do so on a regular basis. I'll just have to brush up on my API calls to implement this.

I have converted much of the code described above into two extension templates, which are available for download under the ClarionMag open source project. The first template, `EmailReport`, can be added to any report you wish to send as an email attachment.

I have included quite a few prompts to allow you to configure the template to your own preferences. You can use your own email procedure or steal Clarion's example as described in the article.

The second template, `EmailViewer`, can be added to a report procedure in an application to create the Viewer as described above.

Simply add the template to an application containing a single report with the Metafile file as the primary Table. The report itself can contain only one detail with an image control. The template will take care of sizing the image to fit the original report. Compile the application, create the file associations and you will have a Viewer capable of printing your email attachments.

Summary

Adding email capabilities in version 5.5 is a nice addition to the Clarion toolset. What is still missing however, is the ability to easily send a report as an email attachment. I have demonstrated in this article one possible, though admittedly limited solution to this problem. There are many ways that this technique can be improved. By submitting the template set to the ClarionMag open source project I am counting on other Clarion developers to take what I have started and improve on it to give us another powerful tool in our arsenal.

Download the template

Reader Comments

Add a comment

My Name Is Tom, And I'll Be Your Server

Published 2001-01-09

It's been a long Christmas break for ClarionMag readers, but the first issue of 2001 is now available on the new Tomcat-powered ClarionMag server. Tomcat is the reference implementation of the JavaServer Pages and servlet APIs, and is available from the Jakarta site.

There are a lot of features I've wanted to add to Clarion Magazine, and now that the magazine is running Tomcat you'll start to see some of these features appear. The first is the weekly ClarionMag reader survey. All the weekly surveys are very short; they're just one multiple choice question. You choose your response, click on the Vote button, and you'll get back a graphic showing the current responses. And the next time you return to the ClarionMag main page you'll see the latest survey results automatically. And remember – the survey changes weekly, so if you intend to vote, do it now!

For the month of January both the new server and the original server will be online. Unfortunately that means there are two authorization systems active, and subscription processing may take a little longer than usual. Thank you for your patience.

There are a couple of points to keep in mind during this transitional month:

- for now, all the 1999 and 2000 PDFs will be available only on the original site (www.clarionmag.com)
- a January PDF will be available at the end of January/beginning of February
- the search engine is only available at present on the original site – it will be migrated to the new server
- the site index is presently only available at the original site, but will be added to this site shortly
- around the end of January, if all goes according to plan, the original server will be taken offline and www.clarionmag.com will point to this server

If you have any questions about how this migration to the new server affects your subscription, please email me at

dharms@clarionmag.com.

Dave Harms
Publisher

Reader Comments

Add a comment

Sending Clarion Reports as Email Attachments (Part 1)

by David Potter

Published 2001-01-09

Part 1 of 2

After Clarion 5.5 went gold, I decided to take another look at a project that had been sitting on my desk for a while. One of the technicians in the factory office had asked if it would be possible to email the daily report to all of the recipients rather than making copies and hand distributing them as is done now.

Previously I had looked at two other solutions. The first was the free product, Hotsend (http://www.hotsend.com). At first this seemed the ideal solution. It didn't require any programming nor did it require any special setup on the client side. However we ran into some problems using it with our particular implementation of Lotus Notes. We were also somewhat put off by the advertising logo attached to the viewer, so decided against this route.

The second solution I looked at was Craig Ransom's freeware Report Archiving and Retrieval (RARS) system (http://www.pcferret.com/files/rars1055.zip). This was an interesting solution but seemed a bit of overkill in our case, as we had no need for archiving the daily report. Included in Craig's system is a very well written set of instructions. After reading these, I saw I could adapt his methodology to better suit my purpose. It is fairly easy to capture the Windows Metafiles created by the Clarion preview class and save them for later viewing. Once these have been saved it's possible to modify a standard Clarion report template and previewer to view and print these saved Metafiles.

Capturing and saving the Metafiles

Capturing the metafiles is simple. First make sure the print preview option is turned on. Then declare the following local variables in any report that you wish to save:

```
!Counter
```

```
Indx            SHORT
!Directory to hold Metafiles
Loc:MetaDir     STRING(128)
!Filename to save metafile
Loc:FileName    STRING(80)
```

Add the following code to the Previewer.Display embed(before the parent call).

```
MetaDir = 'C:\Metafile'
LOOP Indx = 1 TO RECORDS(SELF.ImageQueue)
   GET(SELF.ImageQueue,Indx)
   Loc:FileName=CLIP(Loc:MetaDir)& '\' |
      &  SELF.ImageQueue.Filename     &|
[INSTRING('CLA',SELF.ImageQueue.Filename,1,1)↵
  :LEN(CLIP(SELF.ImageQueue.Filename))-3] &|
      'WMF'
   COPY(SELF.ImageQueue.FileName, Loc:FileName)
END
RETURN(False)
```

SELF.ImageQueue contains a list of the windows metafiles created by the report engine and is used by the print previewer class to display the report image. These files are normally created as temporary files in the c:\windows\temp directory. Once the report has completed, they are then deleted. By copying the files to another directory, I can now save them to play with while developing the program. The RETURN(False) statement will short-circuit the call to the previewer so it will not be called in this instance.

Before proceeding I must have a way to catalog and index the saved metafiles for later printing and viewing. Craig accomplished this in his program by using an ASCII file called report.ini. For my purposes, a standard TopSpeed file can give me what I need. I only need to create a tps file called metafile and add it to the dictionary. The file structure is very simple and needs only two fields. The following structure will do:

```
Metafile   FILE,DRIVER('TOPSPEED'),PRE(MTF),↵
  CREATE,BINDABLE,THREAD
PageKey    KEY(MTF:PageNo),NOCASE,OPT,PRIMARY
Record        RECORD,PRE()
PageNo           SHORT
FileName         STRING(128)
              END
           END
```

I can now create a list of the saved metafiles from the report by including the Metafile table in the list of tables for the report. Then I add the following three lines to the bottom of the loop in PreviewerDisplay embed, after the copy statement, to add the filename and page number to the file.

```
MTF:PageNo = Indx
MTF:FileName = Loc:FileName
ADD(Metafile)
```

The Viewer

Now I need to create a program, similar to the HotSend viewer, that the email recipient can use to view and print the saved metafiles. The easiest way to accomplish this is modify a standard Clarion report, using the following steps:

1. Open the `Metafile` table and loop through the file in page order.
2. Fill a Queue with the metafile list used by the report.
3. Substitute the `PrintPreview` queue with my queue for display.

To do this I only need to create a new application consisting of just a standard report procedure, using Metafile as its primary table. I declare the queue in the LocalData embed of the report:

```
MetaQueue PreviewQueue
```

where `PreviewQueue` is the datatype declared somewhere in the bowels of the class structure, for use with the `Preview` class. Now I can go to the report structure and place the variable `MTF:FileName` in the Detail section of the report so the report will think it has something to print.

The `ThisReport.TakeRecord` embed can now be used to fill the previously declared queue:

```
MetaQueue.FileName = MTF:FILENAME
ADD(MetaQueue)
```

The trick here is to substitute the `MetaQueue` I've created for the `ImageQueue` used by the `PrintPreview` class. In standard Clarion generated code, the `ImageQueue` is passed into the `PrintPreview.init` procedure as a parameter. I can override this, substituting my call to the `Parent.Init` procedure, by placing the following code in the `Previewer.Init` embed (before the parent call).

```
PARENT.Init(MetaQueue)
RETURN
```

The return statement will ensure that the original call to the parent (passing the queue created by the report engine) will never be reached. The previewer will instead be initialized using `MetaQueue`.

After compiling the program, I found the saved report was displayed in the previewer just as I expected. However printing the report did not work as I expected. Instead of the three page report I had previously saved, I get a single page listing all of the

metafiles used for the report. After thinking about it, this made sense to me. The report engine still only sees a line of text and will print it as such. I have to get the report engine to print the image contained in the file, not the filename itself. To accomplish this I use an image control as follows:

1. Go back to the report formatter and delete the header and footer.
2. Expand the detail to fill the entire page.
3. Add an image control and size it slightly smaller than the page
4. At the `ThisReport.TakeRecord` embed add the following code :

```
SETTARGET(REPORT)
?IMAGE1{prop:text} = MTL:FILENAME
SETTARGET()
```

This will assign a new metafile to the image control for each file in the list. This image will now be printed instead of the filename.

Setting the page size and orientation

Further testing brought up another unexpected problem. What if the original report was set to print in landscape mode or on legal paper? With the present setup the report will still print, but the page will be distorted to fit the size of the image as I originally defined it. In order to be able to handle all report sizes it is necessary to:

1. Obtain the original size and orientation of the report and save it so this information can be passed to the previewer.
2. Dynamically change the size of the image, paper size and page orientation in the previewer according to the passed information.

To accomplish these goals it is necessary to :

1. Add two fields to the `Metafile` file structure to account for the paper size and report orientation.
2. Ensure that the necessary equates can be found by including the file `PRNPROP.CLW` in the global embeds of the application containing the reports to be emailed.
3. Obtain the current orientation and page size using the `Report {PROP:LANDSCAPE}` and `{PROPPRINT:PAPER}` properties. This can be done at the beginning of the `Previewer.Display` embed before looping through the image queue.
4. Set the `MTF:PageSize` and `MTF:Orientation` fields during the loop before the `ADD(Metafile)` statement.
5. The Previewer application must now check the above fields and dynamically set the `LANDSCAPE` property for the report. The width and height properties need to be set as well, for the Report, Detail, and Image, to ensure the report prints correctly. This is best accomplished in the

`ThisWindow.OpenReport` embed just after the report has been opened. It is only necessary to read the first record in the `Metafile` to get the necessary information. This can easily be done using the statements:

```
SET(MetaFile)
NEXT(Metafile)
```

Putting it all together

Now that I have all of the individual pieces, it is time to put it all together. The fact that the report will only be distributed internally allows me to cheat a little. If the report had to go to someone who didn't have access to the network it would be necessary to package the metafiles with the list and email the whole package to the recipient. Since in our case all of the recipients are on a LAN/WAN, it is only necessary to place the metafiles in a public directory where everyone can have read access to the files. To accomplish this I just changed the reference of `MetaDir` to a shared network directory.

It would be a waste of resources to send a copy of the viewer with every email so it was decided to put that on the network as well. We have three separate offices all connected by a WAN, so I have to distribute a copy of the viewer/printer to each of the three offices and place it in a local network directory at each office. With this in place, all I have to do is email the Metafile list as an attachment to each address needing the report. As long as the viewer can find the Metafiles in the directory, they can remain in a common location. The problem remains as to how to link the report to the viewer. This is an easy fix and can be done by as follows:

- Rename the .tps file with a unique extension
- Create a file association with the operating system to link the extension to the viewer.

First :

1. Add a global variable in the pathname entry under file properties, in the dictionary, i.e. `!GLO:MetafileName`. This will have the added advantage in that now I am able to create a unique filename for each report generated.
2. Changes must be made to the viewer in order to get the filename from the command line. This only takes a few lines of code at the Global Program Setup Embed in the Previewer application.

```
GLO:Metafilename = COMMAND('1')
IF ~ GLO:Metafilename
  MESSAGE('A valid report file must ' |
    & 'be supplied as a parameter')
RETURN
END
```

The application containing the report must also be changed as well:

3. The defer file opening option must be set for the `Metafile` table in the Global Data|Individual File Overrides section of the application. The Generate file declaration option must be set as well. This will ensure the file is not opened before the global variable `Glo:filename` is set.
4. In the `Previewer.Display` Embed, code must be entered before the loop to set up the name of your Metafile. Here I use the format `PPL#####.PVW`, generating a number using the Clarion `random` function. I have chosen to use the extension `PVW` for this example. You may of course use any unique extension you wish.

```
Glo:MetaFileName = |
  'PPL'& FORMAT(RANDOM(1,99999),@N05)  & '.PVW'
IF Relate:MetaFiles.Open()
MESSAGE(ERROR())
END
```

Note: Be sure to close the file outside of the loop after all of the files have been added.

Now I am able to:

- Run the report several times, generating a unique `PVW` file with each generation.
- Open the metaviewer simply by double clicking on any `PVW` file using Windows Explorer, once I have created a file association with the operating system

That's all for Part 1. Next week I'll show how to email the report.

Reader Comments

Add a comment

Clarion MAGAZINE

Search
Go

Home

COL Archives

Subscribe
New Subs
Renewals

Info
Log In
FAQ
Privacy Policy
Contact Us

Downloads
PDFs
Freebies
Open Source

Site Index

Call for Articles

Implementing SELECT DISTINCT in a TPS Database

by Jon Waterhouse

Published 2001-01-09

Many times the information that is stored in databases is more detailed than the information you want to print out. For example, in a standard accounting system you may store the balance owing on all orders, but what you want to print out is a list of the customers who currently owe you money. On a straight report you'll get duplicate entries when you have a customer with more than one order owing. Or maybe you need a list of the parents of children who had passed through a particular classroom so you can inform them of the just-discovered asbestos hazard, where the parents could have more than one affected child.

Both of these cases would be handled in SQL using a SELECT DISTINCT clause where, in the first case, you would ask for distinct (no duplicates) customers, and in the second, for distinct parents. If you're working with a non-SQL database, you have to write the code yourself.

In SQL you don't really have to worry about how you are provided with the set of records you want. But when you program in a procedural language like Clarion, it is an issue. In some special cases, where the duplicate records are going to be retrieved one after the other, you can write a little filter that throws away records where the current record matches the stored key value of the previous record. But this will not always be the case. In the more complicated situation what you need to do is keep a list of all of the distinct values you have already encountered, and throw away any records where the values match those you have already stored in your list.

The two examples I've already mentioned are based on only one distinct element, but there will be cases where you will want to keep track of two or more elements. One report I wanted to produce the other day was based on eight files. In this report the production of an indicator relies on one or more underlying measures (for example, the indicator for whether people work more after an intervention relies on two measures: the amount they worked before, and the amount they worked afterwards). In turn,

these measures are constructed from fields in data files (derived from client applications and client surveys — we're talking government work here, in case you hadn't guessed), where each field can occur in more than one file. The data files are classified into different types to avoid clutter; there are several very similar client files, several similar employer files, etc.:

```
Indicators
<-->IndMeasLink
  <-<-->Measures
    <-->->MeasFieldLink
      <-<-->Field
        <-->->FileField
          <-->->Datafiles
            <-<-->DataFiletype
```

All I wanted to report was the data file types used for each indicator, i.e. the distinct elements are the indicator ID and the Datafiletype ID. In order to get the report to print with only one entry for each combination of indicator and data file type, all I have to do is set up a queue to store these two elements. For each record I then check to see if the combination of values is stored in the queue. If it is I throw away the record (return record:filtered from validate record), if not I add the values (in sorted order) to the queue.

Creating a filtering template

The code is straightforward to write when required, but it is also something that cries out to be made into a template. Then all you need to do is specify the fields for which you want the printed report details to be distinct. (The completed template is available for download.)

The type of template that is required is an EXTENSION template. This is because it the template must write code into more than one embed point; since you need code both to validate the record, and to make the queue declaration (therefore not a CODE template). The code is not associated with a control, so it won't be a CONTROL template.

The template needs to do three things:

1. Allow the programmer to specify one or more fields to act as DISTINCT restrictions
2. Declare a queue that will hold the set of distinct elements
3. Put code into the `ValidateRecord` embed point.

I've added two other minor bits to improve the functionality of the template:

1. A `#RESTRICT` section makes the template available only for reports
2. The template will report an error in the code generation phase if the programmer neglects to specify any fields.

Programmer input

The programmer input is accomplished using a #BUTTON statement (which opens a new page of prompts), with the MULTI attribute (which allows multiple values to be entered). In this case I only have to specify the field we want to use, so there is only the single #PROMPT statement inside the #BUTTON.

```
#BOXED('Add at least one field'),AT(0,20)
  #BUTTON('&Distinct Fields'),↵
      MULTI(%DistinctFields,%DistinctField)
    #PROMPT('&Distinct Field:',FIELD),%DistinctField
  #ENDBUTTON
#ENDBOXED
```

Each of the entries has to be selected from the field list for the procedure. The programmer would normally be expected to choose from the fields in the VIEW, though it is possible to conceive of a local field (that is calculated based on the retrieved data in each record) being used. The #BOXED statement is for prettification only.

Queue declaration

The most complicated part of the code is writing the QUEUE declaration. Fortunately it is not necessary to start writing this code from scratch. The browse template performs a very similar task in constructing a queue based on the fields that have been added in the listbox formatter and hot field list. This is accomplished by the %ConstructQueue #GROUP in ABBROWSE.TPW. I copied over this #GROUP and modified it.

```
#GROUP(%ConstructDoneQueue),PRESERVE
  #!Modified from ConstructQueue in abbrowse
  #DECLARE(%QFieldName)
  #DECLARE(%QFieldType)
  #DECLARE(%QFieldComment)
  #INSERT(%MakeDeclr,22,%OOPConstruct,'doneq','QUEUE')
%[53]OOPConstruct !Queue declaration for distinct fields
  #FOR(%DistinctFields)
    #SET(%QFieldType, %GetQueueDataType(%DistinctField, ↵
      %QFieldComment))
    #SET(%ValueConstruct,%DistinctField)
    #INSERT(%MakeDeclr,24,%OOPConstruct,%ValueConstruct,↵
      %QFieldType)
    #IF(%QFieldComment)
%[53]OOPConstruct !%QFieldComment
    #ELSE
%[53]OOPConstruct
    #ENDIF
    #!
  #ENDFOR
%[20]NULL END
```

My new version of this GROUP is shorter than the original. I decided

that it was very unlikely that anyone would want to use elements of an array as distinct elements, so I chopped that code out. In addition, the original GROUP expected that the multi-valued token %QueueField, containing the dependent field %QueueFieldAssignment, would contain the list of fields for which the GROUP had to set up the queue. In this template, the list of fields to use is the list entered by the programmer, and these fields are stored in the %DistinctField token dependent on the multi-valued %DistinctFields.

The main part of the functionality of the GROUP is in the #FOR loop, with the calls to %GetQueueDataType. This is where the queue field is declared to be the same type as the matching file (or local data) field. The %QfieldComment token is also filled in by this GROUP to put a comment on the end of the line so that when you look at the code you can see how the type was determined. Mostly the fields the programmer will use are contained in the VIEW, so we expect the queue fields to be commented as " - type derived from field" (the %GetQueueDataType GROUP is in ABGROUPS.TPW).

The other change I made was to hard code the name of the queue to doneq. The original relied on the %ListQueue token for the queue name. The queue has to be placed in the data section; in fact the #AT(%DataSection) segment of the template includes nothing but a call to the %ConstructDoneQueue GROUP.

Validate record code

The code that needs to be put in the validate record embed ultimately needs to look something like this:

```
1  Doneq.fil:field1=fil:field1
2  Doneq.fil:field2=fil:field2
3  Get(doneq,Doneq.fil:field1, Doneq.fil:field2)
4  If errorcode() !record not found
5    Doneq.fil:field1=fil:field1
6    Doneq.fil:field2=fil:field2
7    Add(doneq,Doneq.fil:field1,Doneq.fil:field2)
8  Else
9    Return record:filtered
10  End
```

The first two lines (and lines 5 and 6) can be very easily made with a #FOR statement:

```
#FOR(%DistinctFields)
  doneq.%DistinctField = %DistinctField
#ENDFOR
```

Line 3 requires a comma-delimited list of the fields to be constructed. This is done by building the %ValueConstruct token in a #FOR loop and then outputting the constructed token. The last parts of lines 3 and 7 (the GET and ADD statements), are the same so, although it is not very elegant, the %ValueConstruct token

contains just the list of queue fields, each one prepended by a comma; the remainder of the required text is added as literal strings.

```
#FOR(%DistinctFields)
  #SET(%ValueConstruct,%ValueConstruct & ',doneq.' ↵
     & %DistinctField)
#ENDFOR
get(doneq%ValueConstruct)
```

The whole segment of code looks like this:

```
#AT(%ProcessManagerMethodCodeSection,'ValidateRecord','↵
    (),BYTE'),PRIORITY(4000)
 #FOR(%DistinctFields)
doneq.%DistinctField = %DistinctField
 #ENDFOR
 #CLEAR(%ValueConstruct)
 #FOR(%DistinctFields)
   #SET(%ValueConstruct,%ValueConstruct & ',doneq.' ↵
    & %DistinctField)
 #ENDFOR
get(doneq%ValueConstruct)
if errorcode()
 #FOR(%DistinctFields)
  doneq.%DistinctField = %DistinctField
 #ENDFOR
  add(doneq%ValueConstruct)
 else
  Return Record:Filtered
end
#ENDAT
```

The #AT statement specifies that the code is to be placed in the ValidateRecord method of the ProcessManager at priority 4000 (i.e. before the parent call).

Restricting use of the template

The #RESTRICT section is extremely simple. With this bit of code added the template will only appear as an option to the programmer in a report procedure. The RESTRICT section does not necessarily depend on the %ProcedureTemplate variable (which corresponds to the first parameter in the #PROCEDURE statement). With the advent of ABC, the unit of code generation is mostly the control, rather than the procedure. There is no browse procedure, it is just a window with a browse control, and a form is a window with a save button. Consequently the most common form of restriction is to a type of control, rather than to a procedure.

```
#RESTRICT
  #CASE(Upper(%ProcedureTemplate))
  #OF('REPORT')
    #ACCEPT
  #ELSE
```

```
    #REJECT
  #ENDCASE
#ENDRESTRICT
```

No fields specified

Finally, in the #ATSTART section, I add a line of code that checks to make sure the programmer has added at least one field to the distinct fields list. This will be presented to the programmer at the code generation stage, which should be less confusing than getting "incorrect number of parameter" errors when the compiler encounters GET(doneq). It is also possible to add the #ABORT statement in the same section so that code generation stops when the problem is hit.

```
#IF(%DistinctFields=%Null)
  #ERROR(%Procedure & ' You have not entered ↵
     any fields as distinct elements')
#ENDIF
```

Usage

To use this template just add an INCLUDE statement [#INCLUDE('DISTINCT.TPW')] in any TPL file to bring in the DISTINCT.TPW template file. The template will be available only for reports.

Summary

Writing your own templates to make reuse of existing code easier is often not that hard. For this template, the hardest part was making the queue declaration with the correct data types. I just borrowed the code for this from the browse template. Apart from that, substantially all I had to do was to wrap a few #AT statements around the code I had already written and replace a couple of variable names with the tokens for the fields to be entered by the programmer. I'm sure I will never be as comfortable writing template code as I am writing regular Clarion code, which adds to the time cost of template writing, but for code that is likely to see even moderate reuse, writing a small template is probably the way to go.

Download the template

Jon Waterhouse has been using Clarion since the 2.1 days. His main work is as an economist, and he finds that Clarion is well-suited for applications which impose order on various sets of data. His projects include questionnaire data entry programs, classification software (assigning projects to groups), plus some more interesting scheduling applications. Jon has also used Clarion to link text information together, and is currently developing a program that will store linked snippets of WordPerfect documents and print custom documents composed of several of these snippets. He is currently working for the Newfoundland Government on a project to measure the performance of government employment programs.

Reader Comments

Add a comment

Clarion MAGAZINE

Clarion News

HTML Designer Updated

The latest beta of HTML Designer version 1.0 can now implement HTML Help for any Clarion for Windows application, either ABC or Legacy, from CW 2.003 to C55 Gold. HTML Designer now uses no external LIB or DLL; the templates access the HTML Help API directly providing an easy alternative to Softvelocity's cwHH class. HTML Designer is available from www.clarionshop.com . Current users make sure you get the new passwords and download information from www.Clarionshop.com. This is a major update and most of the functionality should be working. Included in the update are all the files and installations needed to update a development station to use HTML Designer for HTML Help file creation. For any problems or queries regarding HTML Designer, contact bdl@riebens.co.za.

Posted Wednesday, January 31, 2001

RSBackUp Version 1.0 Released

Robert Stanic has released RSBackUp, a standalone EXE backup utility for your applications. Because it's standalone, your user can still do a restore even if they can't run your application because of corrupted data files. Features include: compression using addZIP DLL; automatic calculation of required floppies; large capacity/ZIP drive support; and selective restore

Posted Saturday, January 27, 2001

ZIPFlash 2.0 Released

Sterling Software has released version 2.0 of the ZIPFlash zip

code insertion product. New features include: a template to calculate the distance between two ZIP codes; reverse auto-lookup from a ZIP code - once the ZIP field is accepted the city/state will be inserted; button on map to toggle on/off the major US cities; and auto-insert of longitude/latitude coordinates into a form at the same time the ZIP is looked up. Demo available. Reg. $99, save $50 if you order before January 28.
Posted Friday, January 26, 2001

HTML Help Designer Now Supports Clarion 2003

Riebens Systems has successfully tested the latest build of HTML Designer with Clarion for Windows 2.003. This means that persons coding in any of the Clarion products from Clarion 2.003 to C55ee can implement HTML (chm) Help. This is implemented in 2.003 with the custom HTML Help API interface from Riebens Systems. This build with its associated templates will be available at the end of January 2001 from www.Clarionshop.com.
Posted Monday, January 22, 2001

Gitano Software Office Closed January 22-28

The Gitano Software office will be closed January 22-28, 2001, while Jesus Moreno is on vacation. It's about time, Jesus!
Posted Monday, January 22, 2001

Free PDF Organizer From Gitano Software

Gitano Software has released a PDF reader that lets you organize your Clarion Magazine (and other) PDFs. You can assign categories to articles, search by author, description, category, or free text using the Gitano Software query manager/builder.
Posted Saturday, January 20, 2001

US Postnet Barcodes for Clarion 5.5

The C5.5 release of Postnet Barcodes is available for download. This release includes an addition to the report dialogs to indicate whether the band used includes a USE equate. If a USE equate is used, SETTARGET() is modified to use PROP:Parent. This addition provides better support for use of the barcodes within page headers and footers. This install requires your current PNet password and serial number.

Posted Friday, January 19, 2001

File Explorer Special Ends In Two Weeks

There are only two weeks left to get File Explorer at the reduced price of $69 instead of the normal $99. This price includes free upgrades.

Posted Thursday, January 18, 2001

Imaging Templates Upgrade Available

Version 1.09 of the Imaging Templates for Clarion is now available. Changes include: support for C5.5 (C5 will no longer be actively supported); C3PA approval; Save User options on standalone print and print from image control window; better support for CCS SQL templates; and use of Clarion file dialog instead of OCX file dialog. This release includes various bug fixes as well.

Posted Thursday, January 18, 2001

Solace WordSpell Templates Beta 3 Released

This release fixes problems with running different versions of Word and also removes the irritating office assistant which appears when fields are being spell checked.

Posted Tuesday, January 16, 2001

SocketTools 3.5 Service Pack 3 Released

Catalyst has just released build 3530 (service pack 3) of the SocketTools 3.5 package. This release addresses a number of issues for both the standard and secure editions, and is a free update for those developers using version 3.0 or later. If you're using the Library Edition, the Clarion function declarations have been updated and now include a new set of import libraries (.LIB) for Clarion. In addition to the new secure connection (SSL/TLS) related functions, there are new functions for both the standard and secure editions which allow you to create trace logfiles (similar to how the ActiveX controls worked). This can be a big help with debugging. A new trace level has been added that will log all of the data that is exchanged between the client and server.

Posted Tuesday, January 16, 2001

CapeSoft File Explorer 1.0 Goes Gold

CapeSoft's File Explorer version 1.0 Gold has been released. All registered customers will receive the full install via email. Note that the special price of $69 will expire on Jan 31 2001. After that the normal price of $99 will apply.

Posted Monday, January 15, 2001

Automated Fax Engine for 5.5 Gold Release Available

The AFE gold release for C5.5 is now available for download. This install requires your current AFE password and serial number. Also available are several different pre-compiled AFE server installers for current licensed users of AFE. These also require your current password and serial number. One installer is a completely "silent" install that runs from an INI file containing the install options... no dialogs open within the installer (beta builds only). Another delivers the same dialogs offered within the developer install but without any extra work on your part. There will be two other iterations within the next couple of weeks that provide additional options.

Posted Monday, January 15, 2001

Freeware: The Sterling Data Calculator

The Sterling Data Calculator is a simple popup calculator which uses a BMP file to display the "buttons" - functions include memory store, percentage, square root, logX - as well as the usual add/multiply etc. Demo apps are supplied ready to compile. This Procedure is based on code originally written by Farpoint Software (Cupertino) and modified in 1999 and 2000 by Sterling Data. It is compatible with all versions of Clarion from CW2002 to C5.5 (ABC and Legacy).

Posted Friday, January 12, 2001

Templates Integrate MSWord SpellCheck

Simon Burrows has uploaded the first demo of the new WordSpell templates. These templates allow your application to use Microsoft Word's spell checker to spell check standard text or entry fields as well as SoftVelocity's new RTF control. As long as your customers have Word, you don't need to purchase any third

party OCX's to go with your application. Price is $39.
Posted Thursday, January 11, 2001

Clarion MAGAZINE

Search
Go

Home

COL Archives

Subscribe
New Subs
Renewals

Info
Log In
FAQ
Privacy Policy
Contact Us

Downloads
PDFs
Freebies
Open Source

Site Index

Call for Articles

Look Ma, No Keys!

by Jim Kane

Published 2001-01-16

Recently I received a request for a browse that I had not anticipated when I designed the underlying databases. To my horror, I realized there was no key defined that would produce the required result. Since the application in question was widely distributed, adding a new key would require end-user data conversion; something I fear more than death. On the other hand, I was not sure performance would be adequate with out a key and I knew using the ABC browse template with out a key presents a few problems - like no locator or scroll bar support. Since users and bosses take programmers for granted and expect locators and functional scroll bars on every browse, I knew I need to add support for locators and scroll bars whether there was a key or not. I've been well conditioned.

The first thing I needed to settle was if the performance would be adequate without a key. I started programming toward the end of CPD, my thinking was keys are essential, `Set/Next` is easy to understand, and views are a mystery and an unnecessary complication. But then again I've seen some of those new-fangled SQL back ends provide sub-second responses even when there was not a convenient key.

Unfortunately this project involved TPS files. Since the browse and underlying files would probably only have about two or three thousand records in them, I thought I'd give a keyless TPS browse a try. I started a new application, and wizard-generated a browse. To my surprise, the browse filled with the correct data from across the network quite fast. In fact, if I didn't know for a fact that the browse didn't use a key, I would not have guessed it. To put it mildly, I was quite impressed with the Clarion view engine. Something I thought I knew, that keys were essential, was proving to be false, at least in this case. I should be used to being wrong though. I have two teenage children (or is that two teenage adults) and they point out I am wrong and have 'old' beliefs just about constantly. Why should my professional life be any different?

Well, perhaps there are some doubters out there. Try it for yourself. Take any application you may have handy, or download

the zip file for this article that includes a demo with some sample data. Open the list box for the browse in the window formatter and right click. Select the Action tab. Press the File button, and select the File for the browse, click Edit and then No Key. That deletes the key for the browse. Then, also on the action tab, go down to the Additional Sort Fields entry and type in the order you want. For my demo, I wanted the list sorted by the `KF:Title` field. I also wanted the sort case-insensitive so I typed `Upper(KF:Title)` into the additional sort field entry.

After specifying the sort order recompile, run, and you'll see the view engine has rapidly sorted the data using what ever you type into the additional soft field.

If you did the exercise above or looked at the demo you may have noticed something rather distressing: the scroll bar behavior and locator buttons on the browse's Action tab become disabled when you delete the key. I knew this meant I'd need to write my own locators and scroll bar code. Or was there some way I could still use the ABC classes?

To find out, I did a quick review of how the ABC locator classes and thumb classes work. (The portion of the scroll bar you drag is sometimes called a thumb so here I use the term *thumb classes* to refer to the ABC classes that make scroll bars work.) The locator classes are located in `abbrowse.clw` in the Clarion `\libsrc` directory. Locators take what the user types and finding the record with a value equal or higher that what the user typed.

Locators work on the so-called free element field. If you have key or order with three fields like this:

```
Key
|_field1
|_field2
|_field3
View order: field1, field2, field3
```

and due to a filter or range-limit all the records in the browse have the same value for `field1`, then `field2` is the first field in the key or order that can vary, so it becomes the free element field. A locator used on a browse based on the key above would work on `field2`. Sometimes this same information is expressed in another way. In the example I just cited, the free element field is the second field in the order so the free element position is 2. The ABC `Browse` class method, `GetFreeElementPosition()` would return 2.

As I mentioned, I'm a founding member of the "In God, Keys, Set(), Next() I Trust Club," so I assumed a locator worked by taking whatever the user typed, putting the user input in to the free element of the key, and doing a `set(key,key)/next()` to locate the appropriate record. To my surprise I found out ABC locators do not work that way. In ABC, after an incremental locator processes a keystroke in its `TakeKey` method, it puts whatever the

user typed into the locator into the free element field as one would expect, but then calls the following code:

```
CheckLocator ROUTINE
  IF ~(SELF.Sort.Locator &= NULL)
    IF SELF.Sort.Locator.TakeKey()
      SELF.Reset(SELF.GetFreeElementPosition())
      SELF.ResetQueue(Reset:Done)
```

The `Reset` method, located in the `ViewManager` class in abfile.clw, does this:

```
ViewManager.Reset PROCEDURE(BYTE LocatePos)
  CODE
    SELF.Open
    SET(SELF.View,LocatePos)
    IF ERRORCODE()
```

In the demo keyless browse application, the free element field is the first field in the order. `GetFreeElementPosition()` would correctly return 1 then. The "magic" statement that does the locating in the view is then `SET(SELF.VIEW,1)`. This statement positions the view to the record greater than or equal to what the user typed using a `set(view, order position)` and not using a `set(key,key)`.

Having traced out the code, I was quite relieved. Since the methodology the locator used did not rely on a key, I could use the ABC locator class without writing my own locator code. All the locator class seems to need is knowledge of the which field is the free element field (`KF:Title` in the sample application) so it can put what the user types into it, and specify the position of the free element field within the order (1 in this case).

The thumb classes

The thumb classes are there to keep the browse position and the thumb position coordinated. If someone drags the thumb, the thumbes class reset the browse position. If someone or some code changes the browse position other than by dragging on the thumb, then the thumb classes update the thumb position to correspond to the new browse position.

The operation of the thumb classes is pretty simple. If a user clicks on a record in the browse, the `StepClass GetPercentile()` method is called and returns a number between 1 and 100. That number is then used to set `PROP:VScrollPos` for the vertical scroll bar on the browse.

Likewise, if the user drags the thumb and releases it, the `StepClass GetValue()` method is called, taking as input the value of `PROP:VScrollPos`; this method returns a few characters that can be used to position the browse. Through these two methods, the browse position and vertical scroll position can be kept in

synchronization.

Since the ViewManager Reset method is used to set the browse position in the thumb class just like in the locator class, the thumb class does not need a key. Although both the locator class and the thumb class need to know (have a reference to) the free element field to use the ViewManager Reset method, they each have a different reference to the free element. The locator class is passed the free element in the locator class Init method, while the thumb classes use the free element reference stored in the Sort queue, which is a member of the Browse class. Why two copies of the reference to the free element are stored is a mystery to me, but that is how it is coded. In my case, when I do something silly like that I usually just put a comment in the code that is was done that way for "future flexibility."

It's important to understand how the Browse class uses the Sort queue. Picture the typical browse with a few tabs. As the user clicks on a tab the browse is resorted or filtered to give a different view of the data. For each different view of the data, there is one entry in the Browse class Sort queue. Each record in the Sort queue contains all the information needed to operate the browse for that view of the data. The fields of interest in the Sort queue are:

- a reference to the main key for the view of the data, if any
- the order string for the view the browse is based on
- the filter string for the view the browse is based on
- a reference to the free element field
- a reference to the thumb class to use (if any)
- a reference to the locator class to use (if any)

The Browse class method AddSortOrder(<ThumbClass>,<MainKey>) is responsible for adding a new record to the sort queue and setting the MainKey, free element and thumb class fields. After calling BrowseClass.AddSortOrder() to add a record to the Sort queue, the following Browse class methods of interest can be called to add other information to the Sort queue:

- AddLocator() – add a locator class to the Sort Queue
- AppendOrder() – adds additional sort fields to the order string in the Sort queue

The other odd thing about the Sort queue is the information for the default view of the data (generally tab 1) is added as the last record of the Sort order and the second tab is added as record 1, the third as record 2, and so on. Strange but true! Obviously done that way for future flexibility

You can determine the current order any time you wish for a browse instance BRW1 by using Pointer(BRW1.Sort). As a result of the unusual order of the Sort queue, for any browse with multiple tabs, you'll see code in the ResetSort() method like this:

```
IF CHOICE(?CurrentTab) = 2
  !yup tab 1 is sort queue record 1
  RETURN SELF.SetSort(1,Force)
ELSIF CHOICE(?CurrentTab) = 3
  !tab 3 is Sort Queue record 2
  RETURN SELF.SetSort(2,Force)
ELSE
  !tab 1 is the last record – 3 in this case
  RETURN SELF.SetSort(3,Force)
END
```

The only catch is `BRW1.Sort` is protected so any code that uses it must be inside a `Browse` class method. Once you understand the order the `Sort` queue is built in, the above code almost makes sense.

With the analysis complete, I saw nothing in the operation of the locator or thumb classes that required a key. Both do a `set()` on the view using the free element position number rather than a `set(key,key)/Next()` like my "brought up in DOS" mind expected. Maybe this brave new world isn't so bad after all. Provided it works, that is!

It appeared that to use the existing ABC locator and thumb classes without a key I needed to:

1. Create an instance of the locator class. I chose the incremental locator class in the demo app.
2. Create an instance of the thumb class. I chose the `StepStringClass` since the free element was a string variable.
3. Initialize the two classes created above.
4. Call `AddSortOrder()` to add the thumb to the `Sort` queue
5. Call `AddLocator()` to add the locator class to the browse
6. Set the `FreeElement` field in the Sort queue (`KF:Title` in the demo app)
7. Force the `BrowseClass GetFreeElementPosition` and `GetFreeElementName` methods to return the correct values. (`1` and `KF:TITLE` in the demo app).

Items 1 and 2 are easily accomplished. Any where in the data area, add the following:

```
MyLoc    IncrementalLocatorClass
MyThumb  StepStringClass
```

Now things get a little more interesting. In you look in `ThisWindow.init` you'll see the templates have already generated a call to `AddSortOrder()` that does not have the just-created thumb class as the first parameter. The templates also didn't generate a `AddLocator()` (I've never had good things to say about templates), and since the `Sort` queue is protected, step 6 needs to go inside the `Browse` class. Since I need to correct the

parameters for AddSortOrder(), the embed in the AddSortOrder method before the call to parent looks like a good place to put steps 3 through 6. Here goes:

```
BRW1.AddSortOrder PROCEDURE(,)
ReturnValue          BYTE,AUTO
  CODE
  ! Start of "Browser Method Executable Code Section"
  ! [Priority 2500]
  MyThumb.Init(+ScrollSort:AllowAlpha|
    +ScrollSort:AllowNumeric,|
    ScrollBy:Runtime)
  MyLocator.Init(?KF:title,KF:Title,1,SELF)
  Returnvalue=Parent.AddSortOrder(MyThumb,)
  SELF.AddLocator(MyLocator)
  SELF.sort.freeelement&=KF:Title
  Put(SELF.SORT)
  Return returnvalue
  ! Parent Call
  ReturnValue =PARENT.AddSortOrder(SC,K)
  RETURN ReturnValue
```

Since AddSortOrder is a derived Browse class method, SELF refers to BRW1. I initialized my thumb class to be a runtime distribution using alpha and numerical characters (salt to taste). The incremental locator uses field ?KF:title, the free element is KF:title, it is not case sensitive (that's what the '1' parameter means) and works with browse BRW1. I then call the Browse class AddSortOrder procedure with the correct parameters – a thumb class, and no key. Then I add the locator to the browse since the templates didn't, and set the free element in the sort queue. Since I'm inside a Browse class method, I can act on protected data like the Sort queue without the compiler going berserk. With my work accomplished I then return, never executing the template generated call to PARENT.AddSortOrder.

Step 7 is also straightforward. I don't need a case statement in the demo app since I only have one view of the data (one tab on the browse) but if you have multiple tabs, use code like this:

```
Case pointer(SELF.SORT)
  Of 1
      !code for tab2
  Of 2
     !code for tab 3
  Of records(SELF.Sort)
     !code for tab1
End
```

For my case, before the call to PARENT.GetFreeElementPosition I put this code:

```
Return 1
```

And in just before the call to `PARENT.getFreeElementName` I put the code:

```
Return 'UPPER(KF:TITLE)'
```

With that last bit of code, I compiled and ran. Guess what? No key, good performance for the number of records expected, and all the usual bells and whistles like locators and thumbs working just fine. Maybe keys aren't quite as essential as this old programmer thought.

Download the example files

Jim Kane was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and after graduating served in the US Air Force. He is now retired from the Air Force and writing software for ProDoc Inc., developer of legal document automation systems. In his spare time, he runs a computer consulting service, Productive Software Solutions. He is married to the former Jane Callahan of Cando, North Dakota. Jim and Jane have two children, Thomas and Amy.

Reader Comments

Add a comment

Great Article. For some reason (perhaps because it isn't...