Clarion Magazine

Clarion News

Search the news archive

ClarionMag Free EBook Offer Extended To Friday, April 8

All Clarion Magazine subscribers who subscribe or renew by April 8, 2005 will receive an e-coupon for one free e-book.
Posted Wednesday, April 06, 2005

ClarionMag Adds Three New E-Books

Clarion Magazine has released three new e-books: Learning The Clarion Language, Learning The Clarion Template Language, and Threading In Clarion. Non-subscriber prices are $19.95 each, subscribers $9.95. Subscribers also get free updates.
Posted Wednesday, April 06, 2005

EasyResizeAndSplit 2.08

EasyResizeAndSplit 2.08 is now available. Fixes include: GPF with the RTF control; C61 general fix; C61 iconize and restore modal window caused wrong window size and control positions. This is version for Clarion 5.0, 5.5 and 6.1 (9033). Free for all registered customers. Price: $79
Posted Tuesday, April 05, 2005

SoftVelocity Training On Demand

The first two SoftVelocity courses are ready and will start shipping out within a day or two. Check the outline of lessons that we have available so far, and I think that you will be excited at this product's usefulness to you.

Posted Tuesday, April 05, 2005

In-Memory and IP Driver 2.0 Releases Coming Soon

New version 2.0 releases for the In-Memory and IP driver will be coming your way very soon.
Posted Tuesday, April 05, 2005

EasyCOM2INC 2.01

EasyCOM2INC 2.01 is now available. EasyCOM2INC utility is used to automatically create Clarion include files with the definitions of COM interfaces from IDL files. EasyCOM2INC now comes with EasyCOM Generator to generate needed Classes. Requires Clarion 5.5 or higher. EasyCOM2INC generated class wrapper files require Clarion C6-9033 or higher. Demo version available. Price: $189.
Posted Tuesday, April 05, 2005

DCT2SQL Templates Updated

Modifications to the MySQL template have been made so you can create scripts with the components you need. For example you can create a MySQL script with just table and primary keys for your table creation, and you can create a MySQL script with secondary indexes only for after you do your massive data loads.
Posted Friday, April 01, 2005

cpTracker Enteprise Screen Shots

Screen shots of cpTracker Enterprise (MS SQL) Edition are now available. Pro/TPS version will get the interface improvements as well.
Posted Wednesday, March 30, 2005

ABC Free Templates and Tools Updated

Please note that some of the changes made in this update will require that you revise an object name. You'll know that this is required if you get a compiler error regarding field not found or unknown procedure label. These changes have been made to in order to make the set more standardized around common object template wrappers.
Posted Wednesday, March 30, 2005

xTimers 1.0 CHM Help

The help file for xTimers 1.0 is now available. Also this help file is now included into Demonstration program and Installation kit.
Posted Wednesday, March 30, 2005

Big Scheduler Tamer Template Suite

Over 30 new features have been added to the Big Scheduler Tamer Template Suite; Added support for dynamic week starts on Monday for Calendars, WallCalendars, WeeklyPlanner; Added new control template WeekMonCheckBox for controlling it; Added support for IceTips Previewer; Added built in report for Weekly Planner; Added Conditional Icons and Colors IF statements to Bookings, and schedulers; Added Checkbox to use same icon/color for duration in above; Added New Control template BSTimeLocBar; Added Time Bar Locator support to Bookings Day; Added Time Bar Locator support to Schedulers; Added button effect to Large and Small Calendars; Added code to reduce flicker on Calendars; Added code to handle end of month on Month Ahead and Month Back routines in Calendars; Added embed for CreateTimeColumn Routine to customize/change colors etc in First column in schedulers; Old Static Wall Calendars are no longer supported - switch to dynamic.
Posted Wednesday, March 30, 2005

wPDFControl Wrapper

wPDFControl Wrapper saves Clarion reports to PDF. This release adds the ability to convert existing RTF files to PDF format and save the content of the Clarion RTF Control as PDF (for both Clarion 5.5 and 6.1). Everything is done via the control template so no handcoding is required. Demo available. Upgrade pricing for the wPDFControl owners is also available.
Posted Wednesday, March 30, 2005

Gradient Tool

Create gradients and texts for all you frame/windows. Product features include: Multiple images on frame or any window; Gradient on list boxes and browses, buttons, groups, regions, option groups; Create new gradients on the fly and save the definition for your other apps; Add text to all frames and windows; Change fonts, colors, shadow, orientation, etc.; Extended messages options included - place/size/color/fonts of the buttons; Extended Flat/Transparent control at Global level, included. List price: $ 60.00. C5.x : C6.x, Legacy and ABC.

Posted Wednesday, March 30, 2005

BoTran Released

The latest set of templates and utility applications in the BigTamer(tm) Series is the application translation BoTran(tm) Set. BoTran(tm) also translates field prompts in screens and prompt strings in reports so you can adapt the user terminology in your app to fit the unique terminology for a particular customer. It also translates your applications for different languages. All of the translating is accomplished with a utility application outside of the application itself which obviates the need to have several versions of the same application code for various customers. Just have several translations for each unique customer need. BoTran(tm) has a special introductory price of $149. right now until April 1st when the price will go up to $199.
Posted Wednesday, March 30, 2005

EasyCOM2INC 2.00

EasyCOM2INC 2.00 is now available. Added: New Easy Class Generator (ecg.exe) ver. 1.00; Launching of Easy Class Generator; Method parameters which defined in the IDL as "optional" and has no "default value" now generate as omittable. Changed: New Edit->ClarionRTL procedures menu; TRV file structure changed (so you need to regenerate it again). Fixed: Wrong some dispinterface methods when dispinterface was defined in the IDL file as: [Error! Illegal hyperlink object.] EasyCOM2INC utility is used to automatically creating Clarion include files with the definitions of COM-interfaces from IDL file and now comes with EasyCOM Generator to generate needed classes. EasyCOM2INC's 2.00 price increased from $99 to $189. All customer of EasyCOM2INC 1.xx will get a free update to the new version. (Just download, install ver 2.00 and copy your ecom2inc.ini file from your 1.xx directory into new one.)
Posted Wednesday, March 30, 2005

Icetips Newsletter

The latest Icetips newsletter includes information on the Icetips Previewer and SQL products, new training videos, an update on the status of the MySetups product, and debugging tips. You can also read about a new proposed utility class.
Posted Wednesday, March 30, 2005

BRWAccess Template

Jesmond Spiteri has created three small templates to manage browse security.
Posted Wednesday, March 30, 2005

Prospector Contact Management

Ray Rippey has a contact management demo available. It's temporarily on sale for $99, and is written in C55H. The source is not available. Ray will supply free upgrades to Clarion users who provide feedback.
Posted Wednesday, March 30, 2005

SetupBuilder 5 Developer Edition Build 1078

SetupBuilder is 99% code complete now and documentation will be available next week. This release adds support for billboards, as well as several related improvements.
Posted Wednesday, March 30, 2005

BRT Registration Tamer 1.5

BRT Registration Tamer 1.5 is now available. The template has been changed so you can have the BRT Message Box Window as a local procedure, so you can modify as you wish.
Posted Friday, March 18, 2005

Capesoft SendTo Moves Up A Gear

SendTo enables you to send your data exactly as it appears in the browse to a Printer, File (HTML, Excel, Word, CSV or PDF) or email it directly. Simply drop on the SendTo button on any browse/list in your application, and you have all these options available without having to design extra reports or export procedures. The latest release of SendTo significantly enhances functionality and feature support. For example, both Word and Excel now fully support RGB colors, so the output produced accurately reflects the styles used on the browse (fonts, styles, background and font colors etc.). A number of changes have been made across the board to make the output more consistent and to ensure that the output matches the browse as closely as possible. SendTo requires the following CapeSoft products: Office Inside, NetTalk, Draw and WinEvent. Cost: $99.
Posted Friday, March 18, 2005

AnyFont Goes Gold

AnyFont v1.20 Gold is now more programmer-friendly. In Clarion 6 and above, entering

the global default font is done through the Font Dialog box instead of manually. No more contemplating how to spell your favourite font. The user can now change the character set at runtime too, so people in non-English speaking countries will have no problem choosing Cyrillic or Thai or whatever they need. As per normal procedure, AnyFont will go up to its gold price in the next couple of weeks. Features include: Windows and controls move and resize automatically to allow for the font changes; Window sizing and control moving can be disabled to allow for other modules to perform this function; Can also be disabled at the local level; Ships as source code, not DLLs or black boxes; No other templates needed. Current Cost: $29. New Gold Price effective 22nd March 2005: $49.
Posted Friday, March 18, 2005

BoTran 2.0 Eases Translation

BoTran 2.0 is an easy to use translation template that does not require you to hack the shipping templates. Version 2.0 adds runtime language switching. Supports C4-C6 ABC and Clarion Legacy. $199 USD, introductory price is $149 till April 1, 2005.
Posted Tuesday, March 15, 2005

xTimers 1.0

xTimers 1.0 is a class and template to manage multiple timers in any procedure. Features include: unlimited number of timers; Adding/killing timers at runtime; Extension template for easy setup; Run procedures, post events, execute Clarion code, etc.; Many embed points. No black box (no DLL), only pure Clarion code. Support Single EXE, Multi-DLL (Local Mode, Standalone Mode), 32-bit. For Clarion 6.1 and Clarion 5.5. Demo and install kit available. Full CHM help file will be ready soon.
Posted Tuesday, March 15, 2005

xFText 2.5

xFText 2.5 is now available. New in this version: The ability to use pure Clarion code to write text to the frame. Compatible with Clarion 6.1 (build 9032). Updated documentation, demonstration program and installation kits for Clarion 6.1 (Build 9032), Clarion 5.5 are available.
Posted Tuesday, March 15, 2005

EasyListPrint 1.12

EasyListPrint 1.12 is now available. Additions/changes include: New settings for the header border drawing mode; New settings on the property column - alignment for the header; Redesigned settings on the control/code template. This version is for Clarion 5.0, 5.5 and 6.1 (9033). Free update to registered customers, or $69 to purchase.
Posted Tuesday, March 15, 2005

xPathManager 1.1

New in xPathManager 1.1: Support for Clarion 5 has been dropped; Option to save last opened path and select it when opening SelectPath procedure; Option to reopen SelectPath procedure (shift key or by variable); New two control (Button) templates to Map and UnMap network drives. Updated demonstration program and installation kit for Clarion 6.1 (Build 9032), Clarion 5.5 are available.
Posted Tuesday, March 15, 2005

cpTracker Lite 2.2

cpTracker Lite version 2.2 is now available. Features include: All new Query/Search, Report and Spreadsheet Wizards; Date insert option for task notes; Solution and interface improvements. You can now print multiple lines via the report wizard, override column headings, change fonts, use colors, etc. cpTracker Lite is the task management portion of the full cpTracker, and costs $27.
Posted Tuesday, March 15, 2005

ABC Templates and Tools Updated 02/25/2005

Modified the following class-based templates to use standard object-wrappers: System-wide Hot Keys, Handle Shutdown Requests, Key Sender, ThreadLimit, Lookup Auto-complete an entry field, and Registry. Various other changes/fixes.
Posted Monday, March 07, 2005

BST 3.904

Big Scheduler Tamer Template Suite 3.904 is now available. Includes bugfix in Wall Calendar, Weekly Planner, Booking; you can now use HotField Reget without using the update option.
Posted Monday, March 07, 2005

Clarion Magazine

Converting TPS Files To A New Layout Part 2

by Abe Jimenez

Published 2005-03-04

Last week I described my strategy for converting TPS files to a new layout when upgrading an application, and I showed how to update the file definitions while keeping the old file layouts in the dictionary. Now it's time to convert the data.

The conversion program

It's important that no users are entering data into version 1 of the program while you are converting the data to version 2. You also want to make sure that after the conversion is run all users switch to the new version of the program, or they will be maintaining two separate databases. The conversion program should start off with a screen similar to the one in figure 9.

Figure 9. Conversion warning message

Note that I'm disabling the Continue button until the user checks the box confirming that there is nobody else using the program. I have noticed that when users are installing a new version of an application they tend to just keep hitting the OK or Continue button without reading the window. By disabling the continue button, I make them read the screen and acknowledge that they read my message.

The cancel button on this window simply exits the program if the user is not able to continue with the conversion at this time. The Continue button calls the processes to convert the data.

My first task is to find out if there is any data to convert. I usually run the conversion program at the end of the setup. Most Setup file creation programs let you specify a program to run after all the files are copied. I use Setup Builder from Linder Software, which has this capability. If a database is shared between multiple users, the first one to install does the conversion. The rest of the users will call the program, but it will not do anything. It doesn't even show them the above window.

To manage this conversion process, I need to check if there are any records in the old files, and I need a variable that I will use later to determine if I need to display the window. The Table Schematic Definition for the conversion program's initial procedure should include the files being converted, as shown in Figure 10

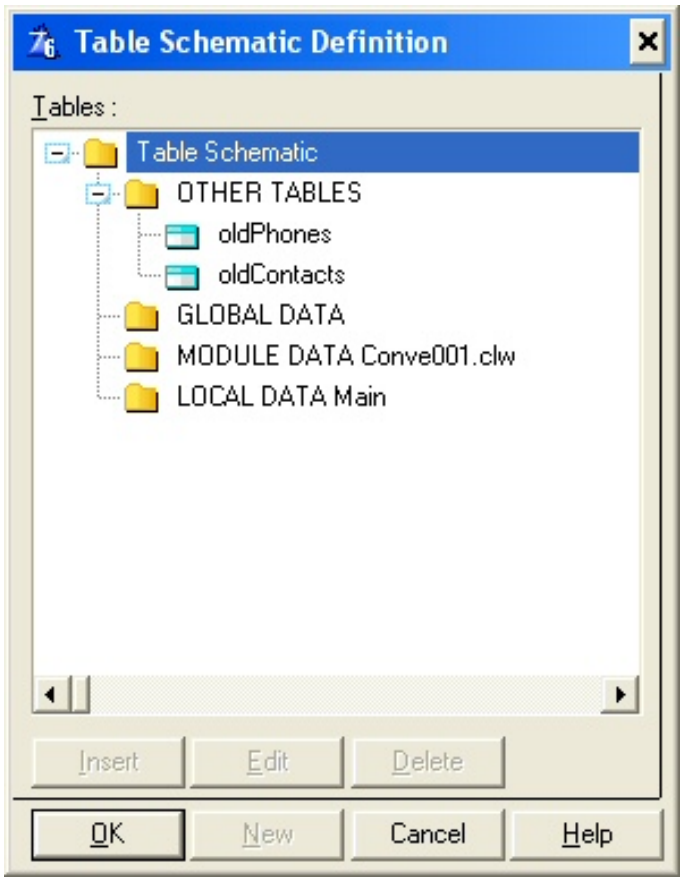

Figure 10. Table schematic for initial procedure

Figure 11 shows the declaration of two local variables. CheckBox is the variable for the user to confirm that there are no other users in the application. The RunConversion variable is used to keep track of whether or not the conversion program needs to run.

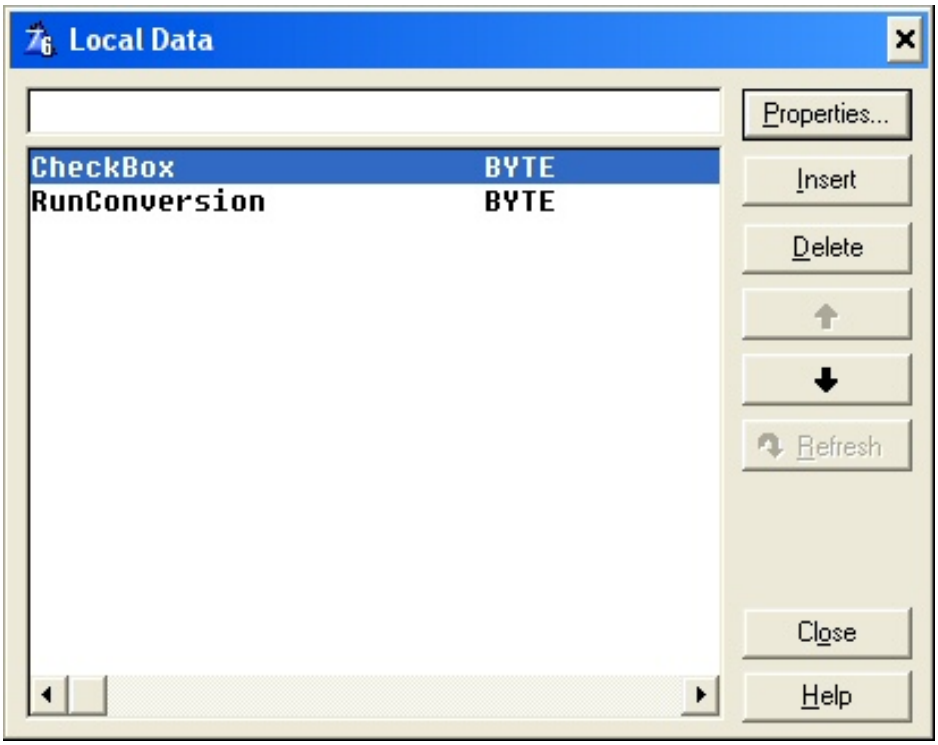

Figure 11. Local variables

This code hides the window until you have determined if it needs to be shown to the user: 0{Prop:Hide} = True

Actually, I also checked the Hide box in the window formatter, so this line of code is not really needed. I only entered it for clarity. Next, I set the RunConversion variable:

```
RunConversion = 0
If Records(oldContacts) then RunConversion = 1.
If Records(oldPhones) then RunConversion = 1.
```

The above three lines initialize the RunConversion variable to false, and then change it to true if any of the old files contain data.

Finally, the following code either displays the window if a conversion needs to run, or closes down the program if the files have already been converted.

```
If RunConversion
   0{Prop:Hide} = False
Else
   post(Event:CloseWindow)
End
```

When the user presses the Continue button on the window, you need to call the conversion processes (described below) for each file that contains data. After the data is copied over to the new files, you want to close down the conversion program. To accomplish this, insert the following section of code in the accepted embed of the continue button.

```
If Records(oldContacts)
   If not Exists(Clip(OLD:Contacts) & '.SAV')
      Copy(Clip(OLD:Contacts),Clip(OLD:Contacts) & '.SAV')
   End
   cvtContacts
End
If Records(oldPhones)
   If not Exists(Clip(OLD:Phones) & '.SAV')
      Copy(Clip(OLD:Phones),Clip(OLD:Phones) & '.SAV')
   End
   cvtPhones
End
Message('The database conversion is complete.',|
  'Conversion Program',Icon:Exclamation)
Post(Event:CloseWindow)
```

I'm checking the number of records in each file before converting it, in case a previous conversion was cancelled after converting some of the files. For example, let's say the program was stopped in the middle of converting the Phones file. When the program is re-run, the Contacts file will be skipped because it no longer has any data in it.

Before I save a backup copy of each file, I'm checking if it already exists. If the program was ended in a prior run, a file may have already been backed up. If you replace the backups, you will lose any data that was already deleted from the version 1 files in the prior run on the conversion program.

The cvtContacts and cvtPhones procedure take care of copying the data. I'll go over these procedures in a bit.

For the above code to work, you need to add these two procedures to the main procedure as shown in Figure 12. If you don't do this, you will get a compile error advising you that the procedures have not been defined.

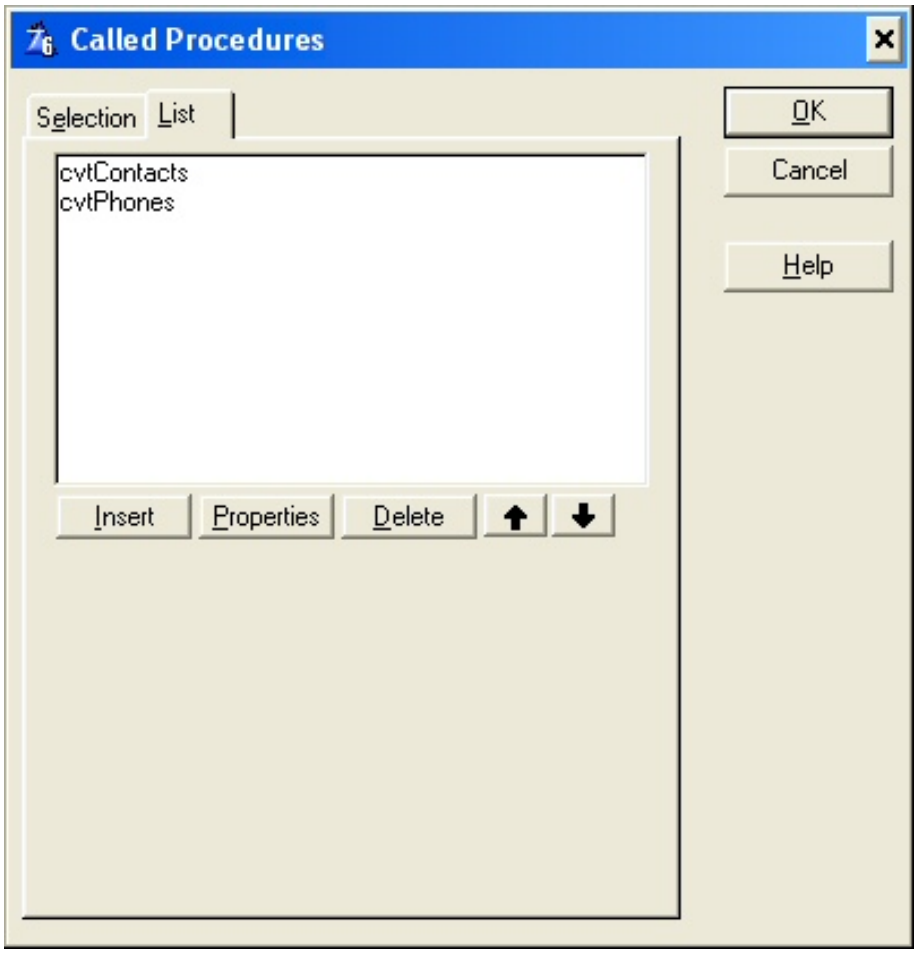

Figure 12. Listing the called procedures

After you close the procedure editor and save your work, the two above procedures will have been added to your application. CvtPhones and CvtContacts are procedures created with the process template. They simply read all records in the old files and write them to the new file. Since both procedures are basically the same, I'm only going to go over the one for the Contact file.

Double click on the procedure and select the process procedure template. You need to enter the old version and new version of the file in the process procedure's table schematic definition as shown in figure 13.

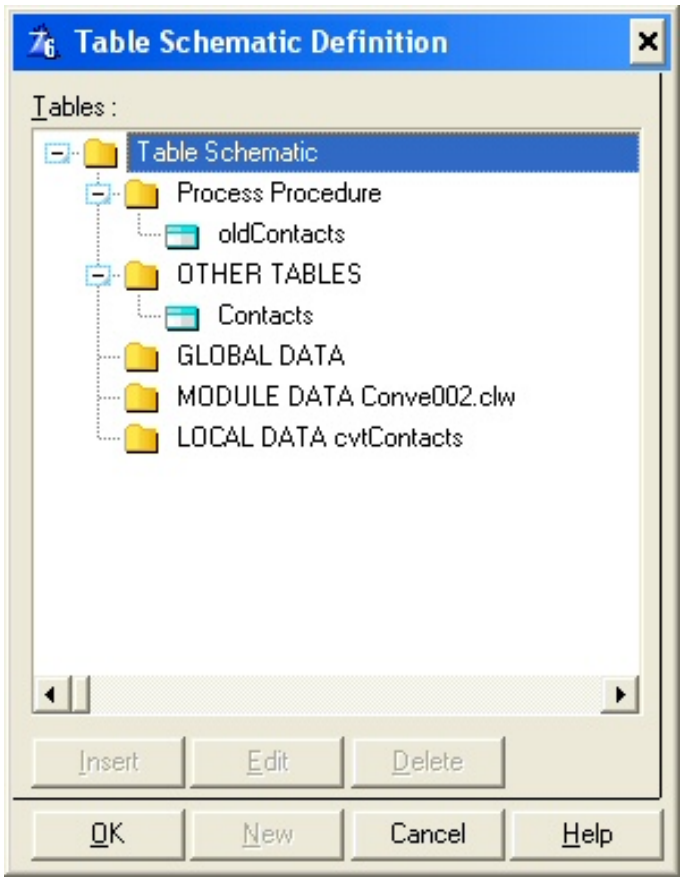

Figure 13. Table schematic for Contact conversion procedure

To copy each record to the new file and delete it from the old one insert the following code in the "Take Record" embed of the process.

```
Clear(CON:Record)           !This initializes the record for the new file
CON:Record :=: CON1:Record  !This assigns the matching columns from the
                            !old file to the new file.
!You can initialize new fields here if you want to give them default values
Add(Contacts)               ! This writes the record in the new file
If not ErrorCode()          ! If the record was added successfully
   Access:OldContacts.DeleteRecord(0)    ! the old record is deleted.
End
```

Note that I'm not using the code below:

```
If Access:Contacts.Insert() = Level:Benign
   Access:OldContacts.DeleteRecord(0)
End
```

The files have auto-incrementing keys. Access:Contacts.Insert() will create new keys and you would lose the relationship between the contacts and the phones.

If you define cascading deletes in your dictionary, you also need to make sure you process the files in the appropriate order. For example, if I had specified that I wanted to cascade deletes in this application, deleting a contact would delete its phone numbers. I would have to convert the phone numbers before converting the

contacts.

You also need to watch out for any triggers you have specified for the files. If you have specified an After-Insert trigger, it will execute. I usually condition my triggers code with a variable so I can stop them from executing. The syntax is as follows:

```
If not GLO:SkipTriggers
   !Put your trigger code here
End
```

The GLO:SkipTriggers global variable above is a byte, which defaults to 0 or false. If I want my program to skip the trigger code, all I have to do is is set this variable to 1.

That's it. Figure 14 shows the version 2 version of the window with the converted data.

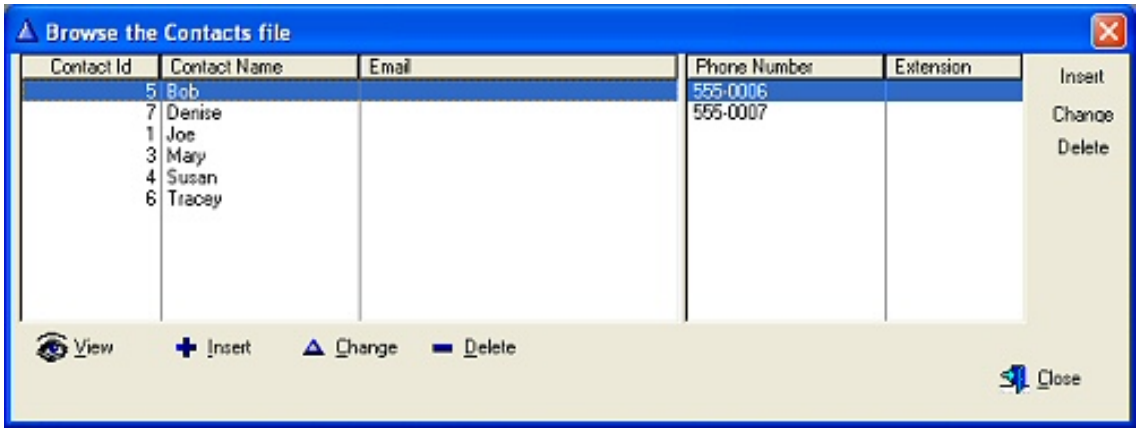

Figure 14. The converted browse (view the full sized image)

Summary

You now have a way to change your database design and distribute the new versions of your application, upgrading data automatically. You will probably want to distribute a document to your users explaining that they need to upgrade all the machines at the same time, because the old version of the program will no longer work after the first user installs the new version.

In the example programs I used Topspeed database files, but this technique should work with other file types as well. I would not recommend this technique for SQL files (even though it should work with minimum changes), because it would not be very efficient. For SQL tables you probably want to run a script to alter the tables.

Download the source

Abe Jimenez started programming in the late 70s in RPG on the IBM System 34. Towards the late 80s he

began using Clarion 2.1 for DOS. Over the years he has programmed in all versions of Clarion, but not continuously. He now develops and markets an application called TechQwest Office (written in Clarion) for the distribution and wholesale vertical markets. His company develops Window, Web and Mobile Device applications in Clarion, VisualStudio.net and Java.

Reader Comments

Add a comment

- » Thanks for giving me some good ideas. I am doing a small...
- » Don, Memo's are stored in another file, so you would...
- » Thanks, you have been a great help.
- » One way I make sure no one is running the program is to...
- » Carl, If I understand you correctly, this will keep oher...

Clarion Magazine

Enriching The User's Experience With RTF Displays

by Stephen Bottomley

Published 2005-03-11

I recently completed a project in which I used the Clarion RTF control to create a customized display that offered better formatting than the usual form or list/browse box. RTF files are just text files, and with a little knowledge, which I'll reveal here, you can easily create your own customized RTF to meet your specialized display requirements, whether it be on the fly as I show here, or saved to files for later use.

Let me start by saying that I'm not an expert on RTF. What little I've used here, I originally created by using Wordpad to generate the skeleton layout of an entry I wanted to display. I then cut and pasted the RTF code into my app, as needed

The scenario: "We would like a library catalogue. We don't need anything fancy and our library is too small to make purchasing an off the shelf system viable. Each entry only has about a dozen fields."

No worries, how hard can it be? Take one of their catalogue entries, build a dictionary based on the fields it contains, and voila. "Here you go, how's this?"

"It's good but not quite what we were thinking." (Uh oh, I've been here before.)

"We'd like to be able to add keywords and abstracts and group information and search it. When we search it we would like to be able to see where in the entry the hit occurred. Oh, and by the way, not every entry has the same number and type or order of fields. Monographs have x and y and... and videos have x and z and... and maps have..."

The scenario revisited: "We would like a library catalogue. We want as many bells and whistles as we can get, but we don't want to pay. We want to be able to create layouts for different entry types with fields that can contain any data types in any order. Some of the fields will contain quite a bit of text. We want to be able to easily read the entries and see where search hits occur. The information in the catalogue is going to be used in several applications so we need to store the raw information only for each field."

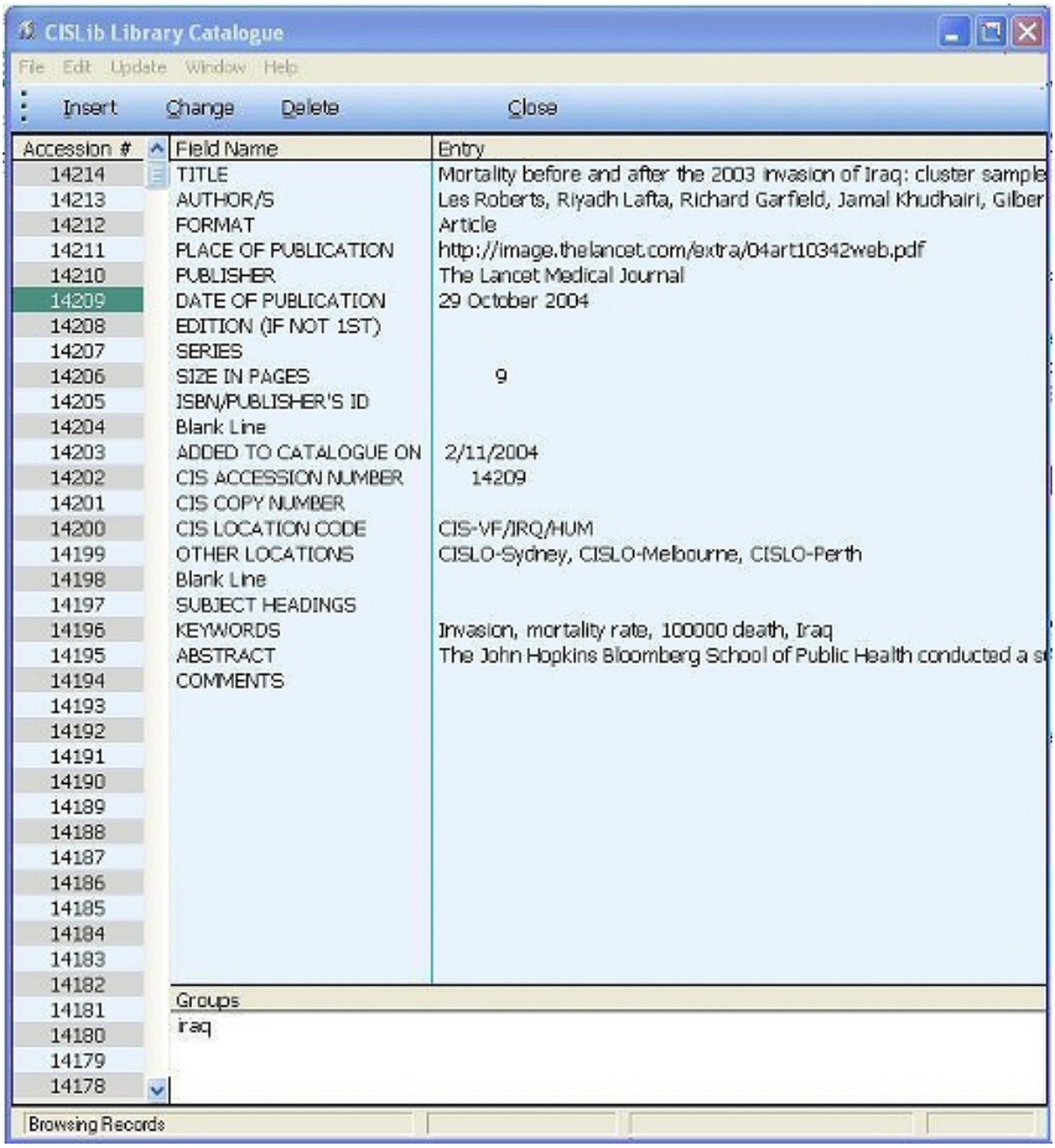

Figure 1. Using a standard browse technique (view full size image)

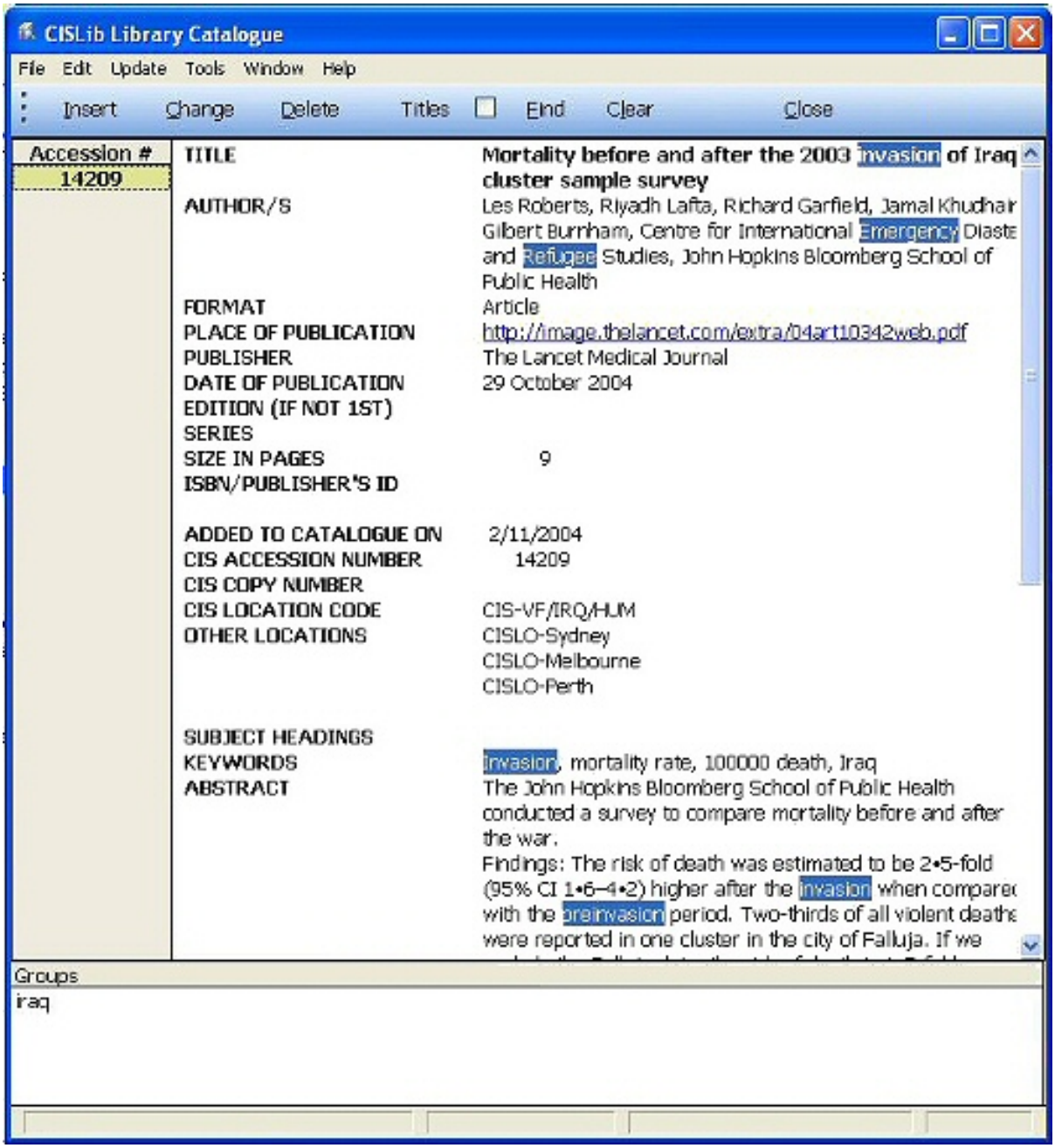

Figure 2. Using an RTF control (view full size image)

Consider Figures 1 and 2 above. Both are of the same application but using two different techniques for delivering information to the user. The first uses a standard browse template in a one-to-many relationship and with no way to provide text wrapping or hit highlighting as required. The second uses the Clarion RTF control.

The solution

To begin with, I replace the old field browse box with the RTF Text Control templates and make sure the RTF box is checked. I set the Value Mode to Field then rename the Rtf class to something useful. In this case I called it `RtfControl`. If you allow control resizing you will need to alter those

parameters as well.

For displaying the RTF, all the work is done in the `TakeNewSelection` embed for the Accession # browse, which runs down the left hand side of both Figures.

To hold the RTF that is going to be displayed I have a `CSTRING` variable labelled (funnily enough) `Rtf`.

If the user has entered a query, the Accession # browse is limited to only those records that contain fields with hits, and those hits will be highlighted in the display. A query can be inactive or active, which allows the user to toggle between displaying all records or only those records with hits without re-entering the query. If a query is active, and in order to add the highlighting, I need to remember where each field's segment of RTF starts within the Rtf string. For this I have a variable called `StartSlice`.

Creating the RTF display string

To create the string that holds the RTF to be displayed, I first add the RTF header information that is mandatory for any RTF display. It contains font and colour tables, which do much the same thing as a Clarion `EQUATE`, and then I add the paragraph format that will be the same for every entry. I won't go into any detailed explanation of RTF syntax; if you're interested, you can read the RTF 1.6 specification at MSDN.

In my code I have two font choices, Tahoma and Arial, and two colour choices, white and blue that will be used to highlight the query hits. I also have a paragraph setting with the line indent and tab stop (`\li` and `\tx`) set at 2840 twips (a twip is a TWentIeth of a Point, or 1/1440 inch); the first indent (`\fi`) set to negative 2840 (for a two inch outdent) and the right border (`\ri`) set to 64 twips so the text doesn't run up against the right hand edge of the RTF control.

At the same time if a query is active, I initialize the StartSlice variable.

```
Rtf = '{{\rtf1\ansi\ansicpg1252\deff0\deflang1033' & |
      '{{\fonttbl{{\f0\fswiss\fprq2\fcharset0 Tahoma;}' & |
      '{{\f1\fswiss\fcharset0 Arial;}}' & |
      '{{\colortbl ;\red255\green255\blue255;' & |
      '\red49\green106\blue197;} \viewkind4\uc1\pard\' & |
      'fi-2840\li2840\ri64\tx2840\b\f0\fs20 '
if QueryActive
  StartSlice = 1
end
```

Next I loop through the fields for a catalogue entry and build up the RTF:

```
FLD:CAT:SysId = CAT:SysId
clear(FLD:SortOrder)
set(FLD:Key_CAT:SysId_SortOrder,FLD:Key_CAT:SysId_SortOrder)
loop until Access:Fields.Next()
  if FLD:CAT:SysId <> CAT:SysId
    break
  end
```

For each field I retrieve information from the field type file. This contains the label for the field, the data type and any field formatting information. I build the RTF so the field label is in bold using the default font, and I set the font size (`\b \f0 \fs`). I then add the field label, turn the bolding off, and insert a tab (`\b0 \tab`) to create the indent.

```
FT:SysId = FLD:FT:SysId
Access:FieldTypes.Fetch(FT:Key_SysId)
Rtf = Rtf & '\b\f0\fs20 ' & clip(FT:FieldName) & '\b0\tab'
```

If the field type is a title (`FT:SysId` of 2) for the entry I want to make it bold as well.

```
if FLD:FT:SysId = 2
  Rtf = Rtf & '\b'
End
Rtf = Rtf & ' '
```

Notice that RTF likes a space after processing instructions and before any text.

Next I determine if the field has any special formatting (like dates and numbers etc.) and format accordingly.

```
if FT:Format
   Rtf = Rtf & clip(format(clip(FLD:Entry),FT:Format))
 else
   Rtf = Rtf & clip(FLD:Entry)
 end
```

If the field was a title I have to end the bolding.

```
if FLD:FT:SysId = 2
  Rtf = Rtf & '\b0'
end
```

And last of all, end the line.

```
Rtf = Rtf & '\par '
```

If there is an active query,the ParseQuery routine is called (see below) to go back over the section of

RTF for the current field that has just been created and add hit highlighters for any results.

```
    if QueryActive
      do ParseQuery
    end
  end
```

Before finishing I replace any carriage return/line feeds in the database text with line breaks and tabs to give the proper indenting when displayed as RTF.

```
  StartSlice = 1
  loop
    StartSlice = instring('<13,10>',Rtf,1, StartSlice)
    if not StartSlice
      break
    end
    Rtf = Rtf[1 : StartSlice - 1] & '\par\tab ' & |
          Rtf[StartSlice + 2 : len(Rtf)]
  end
```

Lastly I terminate the RTF stream, clear any old RTF from the control and load the new RTF into it. The first `SELECT` call forces the RTF control to scroll to the top if the entry is longer than the screen; the second reselects the Accession # browse box to enable the user to keep scrolling down the list.

```
  Rtf = Rtf & '}<0>'
  RtfControl.TakeAction(RTFToolbar:CtlButtonNew)
  RtfControl.SetText(Rtf)
  select(?RTFControl,1)
  select(?ListAccessionNumbers)
```

The ParseQuery routine

The query return list is a queue (`SearchTermQ`) built by the search engine that contains an element for every hit in the order in which they appear. Each element is made up of the catalogue entry's sysid and the word/phrase that makes up the hit.

The `ParseQuery` routine checks the query return list to see if the current field has any matches. If not, I set the `StartSlice` position for the next field to the length of the RTF string. If there is at least one match then `StartSlice` and `INSTRING` are used for each iteration of the query return list for the current field to locate and highlight the hit words it contains:

```
    loop Found = 1 to records(SearchTermQ)
      get(SearchTermQ,Found)
      if SearchTermQ.CatSysId = CAT:SysId
```

```
      break
    end
  end
  if not (SearchTermQ.CatSysId = CAT:SysId and |
          SearchTermQ.FieldTypeId = FLD:FT:SysId)
    StartSlice = len(Rtf)
    exit
  end
```

I then loop through the query result queue for the current field and locate the corresponding words in the new segment of the RTF stream.

```
  Counter = Found
  loop
    get(SearchTermQ,Counter)
    if SearchTermQ.CatSysId <> CAT:SysId or |
        Counter > records(SearchTermQ) or |
        SearchTermQ.FieldTypeId <> FLD:FT:SysId
      break
    end
    loop
      StartSlice = instring(SearchTermQ.Words,Rtf,1,StartTerm)
      if not StartSlice
        break
      end
```

I have to make sure I have the right word. For instance, if the result word is refuge but I've actually found refugees, then I need to keep looking. With the results from the search engine I only need to check that the characters before and after fall within valid bounds. Because I have a space after each RTF processing instruction, and the backslash that precedes instructions falls outside the valid bounds, I don't need to worry about any extra checking.

```
      if inrange(val(lower(Rtf[StartSlice - 1])),97,122) |
            or inrange(val(lower(Rtf[StartSlice - 1])),48,57)  |
            or inrange(val(lower(Rtf[StartSlice +  |
            len(SearchTermQ.Words)])),97,122)  |
            or inrange(val(lower(Rtf[StartSlice + |
            len(SearchTermQ.Words)])),48,57)
      StartSlice += len(SearchTermQ.Words)
        cycle
      end
      break
    end
```

If I find the word or phrase the result queue is referring to, I splice in the RTF to change the text to white and the background to blue (`\cf1 \highligh2`) before the term and then back to the default at the end of the term (`\cf0 \highlight0`). I also adjust the start point for the next iteration by

setting StartSlice to the position of the found term plus the length of the term plus the additional 32 characters of highlighting code.

```
      if StartSlice
        Rtf = Rtf[1 : StartSlice - 1] & '\cf1\highlight2 ' & |
          Rtf[StartSlice: StartSlice + len(SearchTermQ.Words) - 1] & |
          '\cf0\highlight0 ' & Rtf[StartSlice |
          + len(SearchTermQ.Words) : len(Rtf)]
        StartSlice += len(SearchTermQ.Words) + 32
        Found += 1
        get(SearchTermQ,Found)
      end
      Counter += 1
    end
```

Last of all, I set the `StartSlice` variable to what will be the beginning of the next fields segment in the RTF stream.

```
StartSLice = len(Rtf)
```

Using this method has several other advantages over the Clarion list box. On versions of Windows that support it, a user can zoom in and out of the displayed entry by holding down the control key and using the mouse scroll wheel. It also makes it very easy to re-purpose the display into an infinite number of variations like just listing only titles for a quick scroll through the catalogue, as Figure 3 shows.

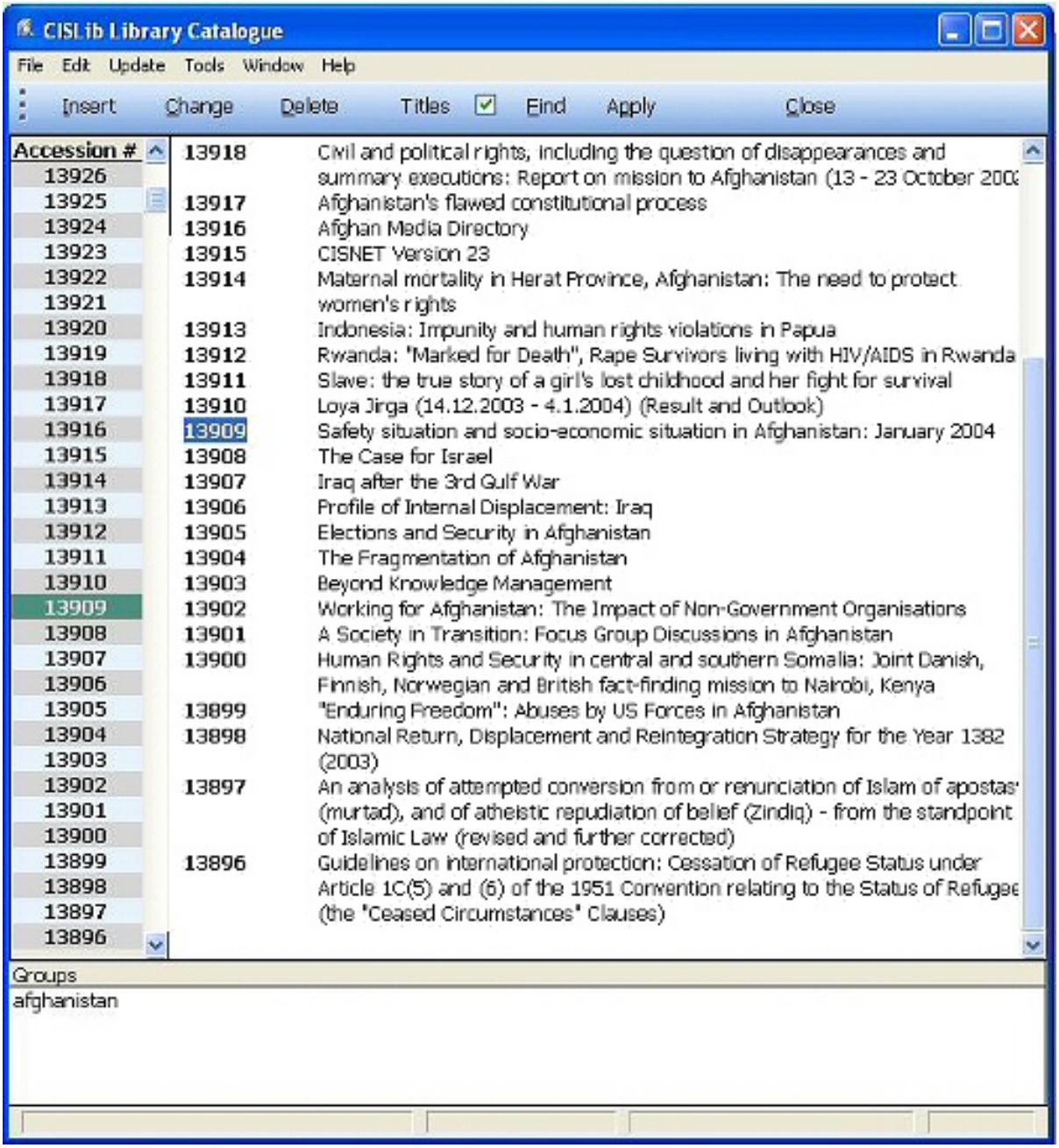

Figure 3. Listing titles only using the RTF control (view full size image)

In Summary

Text is becoming more and more of a requirement for applications, and large text fields have never displayed comfortably within the Clarion list box. What I have shown here is one very simple example of how to overcome that limitation, and I am sure that with your applications you can come up with many more ways of making the end user's experience just a little bit richer.

Download the source

Steve Bottomley is a long time user of Clarion, a member of Team Topspeed, and works for the Australian Government.

Reader Comments

Add a comment

- » A nice use for the RTF control I had not thought...

Clarion Magazine

TRANSCRIPT: Planet Clarion for Friday, February 25, 2005

Published 2005-03-14

Dave Harms: This is Planet Clarion for Friday, February 25th. I'm Dave Harms, good day. Well, as you can probably tell, I am temporarily without my cohost, Andrew Guidroz. Andrew will be along in a couple of minutes and today we're going to be playing a conversation with had with Soft Velocity president, Bob Zaunere. Bob has a lot to say about the upcoming next version of Clarion, as well as Clarion.NET, so I think you'll definitely want to listen that. A couple of points to take care of first. Several listeners have wanted to know where we got our theme music. The song is called Two Tone Blues, it's by an LA studio musician who calls himself and his band Guitar Boy. You can find them at guitarboy.com and you can also download the song Two Tone Blues at garageband.com, which is where we got it.

I'm going to take off my Planet Clarion podcaster hat for a second and put on my Clarion Magazine publisher hat. A few things I wanted to tell you about the magazine. First, if you're getting this broadcast via the RSSV, then you haven't been to the ClarionMag website in the last week or so. Head over to clarionmag.com and have a look. We've done a major overhaul of the website. I think it's a cleaner look and you should find it easier to fin information, especially on the homepage. Separate sections there now for the latest free content, the latest subscriber content. The news items are a little more prominent again. Fonts are now fully scaleable in Internet Explorer, not just in FireFox.

And there is also a My ClarionMag link on every page, and that will take you to a page where you can review your account information. On the homepage, if you looked below the subscriber content listing, you will see a couple of paragraphs on our print books and on something new, a line of e-books which we're launching in March. We are going to continue with the print books, of course, but the idea behind the e-books is to make it

easier for you to find what you're looking for on the ClarionMag website. So these will be PDF collections of articles for the most part, focused on specific programming topics. These will be generally priced between nine ninety-five and nineteen ninety-five. There will be discounts for subscribers.

And we're looking to launch the first e-books in March. And to kick things off and also by way of celebrating the start of our seventh year in publication, we are offering a coupon, actually an e-coupon, for two free e-books. So if you subscribe or renew by March 5th, 2005, the e-book will be added to your account and you can view your e-coupons on your ClarionMag page. And if you're a subscriber, you also get free updates to e-books for as long as you have an active subscription. I think you're going to like these e-books a lot. There's some really cool stuff that will be coming out. So head over to clarionmag.com and check the homepage for more details, and if you have a subscription, log in to your My ClarionMag page to see your subscription status.

And also make sure that we have your current e-mail address and that you're signed up for subscription renewal reminders. This is fairly important, because Clarion Magazine subscriptions now include all the back issues and if you renew before your subscription expires, you do pay a significantly lower rate for the renewal.

Okay, it's time to put my Planet Clarion podcaster hat back on. As I said today, we're playing a conversation we had with Soft Velocity's Bob Zaunere, and during this conversation, Bob refers several times to a conversation that he and I had been having earlier, and that's because Andrew wasn't able to join us right away. I do expect we'll play a bit of that earlier conversation on the next podcast, but right now, let's pick it up where Andrew joins in. <tape plays> Andrew, you there? I heard a click. Good day, Mr. Guidroz.

Andrew Guidroz: Good day. Hey, cool

Dave Harms: Fresh in from the hustings. Do you call it the hustings down there, Andrew?

Andrew Guidroz: Pardon me? Hustings?

Dave Harms: I guess you don't. The hustings. When you're out on campaign, you're out on the hustings.

Andrew Guidroz: Oh, okay. No, we don't call it that over here. Here we call it working the graveyard because… Well, never mind. I just drove in four hours from Biloxi, Mississpi, from the Beau Rivage, so I'm kind of disoriented here. I walk in, my phone is

ringing in my office. I'm like, "What is going on?"

Dave Harms: Yeah, well you missed about an hour of really cool stuff. I wasn't going to – actually, we were just talking at first and I thought, "You know, I better record this because now Bob's going to tell me all this stuff and I'm going to forget it and we're going to want to talk about it again later, but we've already probably going to be covering it." Just to get you up to speed… First of all, say hi, Bob.

Andrew Guidroz: Good morning, Bob.

Bob Zaunere: Good morning, Andrew.

Andrew Guidroz: I'm using a terrible microphone. Somehow or another I broke the mic on my headset, so I'm using a handheld. I look like a Star Search wannabe. No, what is that thing now? American Idol. Someone's about to tell me that I'm not cut out for this type of thing. I can see it.

Dave Harms: So Bob's been telling me about the new IDE. I started out by saying, "Bob, tell me about Clarion 7."

Andrew Guidroz: Hold one second, I've got a call. I'm sorry.

Dave Harms: Oh, you going to do this to me? You can't do this to me. He always does this. It's like, "One second." He's got to answer the phone and he's got to tell somebody to go pick up the bag of money from this place and take it over to that place. This happens a lot.

Bob Zaunere: He's a busy guy. Is Andrew kind of like the campaign manager?

Dave Harms: He's like the campaign manager, yeah. He's basically running the campaign.

Bob Zaunere: Well, very heady stuff.

Dave Harms: Very heady stuff, very busy guy, and he's back.

Andrew Guidroz: Who, me?

Dave Harms: Yeah. No, me.

Andrew Guidroz: You know, I had all that and I had a bunch of my clients over. We had a big Super Bowl party, so I had to go do that trip this week.

Dave Harms: You sound a little like you've had some… You sound a little worn out. Long weekend?

Andrew Guidroz: I came down with a head cold, and of course, you don't ever go to sleep when you go to these things so that kind of doesn't help either. You know, the casino has no clocks up on the wall, so you just stay up on me for second now. Isn't this great? Don't you love this, Bob? Live radio.

Bob Zaunere: You know what's great too though? Is we can do this and it doesn't cost anybody anything. So it's no problem. Yeah, so anyways, Andrew…

Andrew Guidroz: Yeah? Go ahead.

Bob Zaunere: No, go ahead. I was going to say we had some interesting conversations.

Dave Harms: That was the famous rubber boots. That was my famous rubber boots. I'm actually wearing them, because when Bob called, I was just on my way out to go to the bank and <overlapping talk>. Bob's given me other stuff to take to the bank, so it's okay.

Andrew Guidroz: Oh, really?

Dave Harms: You want to hear about this new Clarion 7 IDE? You know what, it's not even a Clarion 7 IDE.

Andrew Guidroz: Well, what is it called? It's the new Clarion IDE, independent of 7 and net, right?

Bob Zaunere: How did you know that? Has Dave sent you transcripts already?

Andrew Guidroz: No.

Bob Zaunere: You know, actually it does need a name, Andrew. And now to find out that you're in the campaign business and you've got that type of, you're obviously good at that type of thing.

Dave Harms: He's done some signs, I know that.

Andrew Guidroz: I think John Smith should be the name – something generic, something that doesn't have any connotations to it.

Dave Harms: Okay, you've been running into name connotation problems with your candidate? We won't go there, never mind.

Andrew Guidroz: My candidate had to change his first name for the election. I mean, this was definitely interesting.

Dave Harms: You know, I guess I was actually just looking today at that April Fool's piece that I wrote a couple of years ago about Soft Velocity Bob, so maybe you want to call the new IDE Bob.

Andrew Guidroz: I think Bob would work.

Bob Zaunere: I don't think we're going down that path.

Andrew Guidroz: So I was correct? Go ahead and tell us about it, Bob. I mean, was I right that it's a unified IDE for all the Clarion languages?

Bob Zaunere: For all the Clarion versions – more specifically, Clarion versions in languages.

Dave Harms: Versions and languages.

Bob Zaunere: If we were to consider the .NET a separate Clarion language, which I really don't, but one could say that. So essentially what I told Andrew about is what…

Dave Harms: You mean Dave.

Bob Zaunere: Yeah, Dave. Sorry. See, we've been talking for an hour and a half and I'm working. I'm starting to lose track now. Here's what we've got, Andrew. We've got an IDE that will let you work with any version of Clarion back to Clarion 2.0…

Dave Harms: Wait, wait, wait. Clarion for Windows 2.0.

Bob Zaunere: Thank you, Dave. Yes.

Andrew Guidroz: Wait, wait. An IDE that works with any version of Clarion? I'm not sure what that implies.

Bob Zaunere: See, Dave? What'd I tell you? Am I right? Here's what it implies and here's some of the reasoning behind it – we know that there are people out there, many, many, probably most in fact, who have applications that they have built in prior versions of Clarion – Clarion 4, Clarion 5.

Andrew Guidroz: Oh, man. Wait, wait, wait. I was behind you, now I'm ahead of you.

Bob Zaunere: You think?

Dave Harms: You think you're ahead anyway?

Andrew Guidroz: So the call I got last week from a client that said, "Hey, there was an old app you worked for us many years ago that was under Clarion 2.0", actually 2003 I guess would be the technical number, <inaudible> all those old libraries and all those old DLLs and everything, and the client says, "We need to go forward with this thing. But we bought personally a version of 5.5 and we're not sure we can migrate forward. What, what, what?"

Dave Harms: I thought you were ahead? It sounds to me like you're floundering a little bit there, Andrew.

Andrew Guidroz: What I'm starting to think is if you got the new IDE and you got the old DLLs, you trying to tell me that you could not bring it forward? You don't have to convert the app to the newer version using the new IDE?

Bob Zaunere: Light bulb, yes. You still there?

Andrew Guidroz: Yeah, I'm there.

Bob Zaunere: I heard a funny sound like <noise>.

Andrew Guidroz: Yeah, I fell. That's what it was.

Bob Zaunere: What it means, Andrew, is that you can fire up your new IDE, whatever name it may have, and having pre-registered 2 is one of the available environments, you go ahead and open up and work with that app, correct bugs, do whatever you might need

to do, add a new feature, change a database, whatever work you have to do in it, use the new IDE, take advantage of all the features it has in it, do a build, be guaranteed that you have used the run times, the compiler, the lib source files, the templates, everything.

Dave Harms: Third party libraries.

Bob Zaunere: Third party libraries, DLLs, anything. So essentially it means you've got a zero migration path to take advantage of the new IDE.

Andrew Guidroz: Can I send the app back to the other developer who's working on it with me, who isn't upgraded? He's still using the old 2003, say. Is that possible?

Bob Zaunere: You can as a TXA. The app format obviously is going to change. Well, maybe not obviously. But app formats always change, DCT formats always change. But it isn't really a problem because of what I said. So if you need to send it back, you say, "Here's a TXA. Just fire up Clarion and import it and you're happy and you've got all my changes, and I'm happy because I didn't have to go back in and find my way around the od IDE. I've got to take advantage of new features like solution based project system, and the new editor, and the new dictionary, and all that good stuff."

Andrew Guidroz: So you're saying the old templates will have the new look, the look of the new IDE?

Bob Zaunere: The old templates will have retained all of their same prompts, they will be identical. However, yes, they will take on the look, and feel, and capabilities of the new IDE. In this case, what I'm referring to is the flattening of the IDE, the ability to have multiple windows open, and to do things you just can't do right now. It wasn't a specially trivial thing to do.

Dave Harms: I can imagine, yes.

Bob Zaunere: But it has, in my opinion, great value. And Andrew, interestingly enough, we had this conversation and I think we probably went into more detail than I just went into right now, but Dave said, "You know, maybe I should put up a survey on the website and see what people think of it." And I said, "You know what? I think a lot of people are going to go, look at it, if the question said, "Would you see a benefit if the new IDE could support multiple versions of Clarion?", and they would scratch their head and go, "I don't know what that means. No, I don't see any benefit." But from my perspective, it has a huge benefit from a techno point of view, for the obvious reasons. From a marketing point

of view, and I said to Dave, equally so because it means that you have zero migration. If you have that app you worked them for the last five years on, you want to take advantage of the new IDE. You don't have to say, "Ah, now I have to allocated two, three months to port and test everything to use the new run time and the new compiler and all that."

Dave Harms: And it's really a whole separate product, so now you have the IDE as a product and you have the version of Clarion as a product, and you have Clarion.NET as a product, and you can have…

Andrew Guidroz: You're actually going to sell…? Wait, now I'm totally… You're going to sell this thing separate is what you're thinking now?

Bob Zaunere: That is what I told Dave was an option, that we could very well do that.

Andrew Guidroz: How would someone buy…? I'm not sure if I understand. How would they buy 7 without the IDE?

Bob Zaunere: Well, that's why I said I may not call it 7, but it would be called the Clarion 2005, whatever it might be called. So how would you do it?

Andrew Guidroz: So you would unbundle the IDE from the actual compiler is where you would go with this?

Bob Zaunere: If you think about it… Well, maybe you don't. I'll explain it to you. Think about it this way – the IDE is an application. The fact that it can communicate to the compiler and knows about database drivers and the correct versions to link in and the correct version of the RTL to link in, etc., is not the application. That's just properties of the app gen, if you will.

Andrew Guidroz: Sure.

Dave Harms: But now the app gen, if I understand you, or the environment, what we used to think of as the environment, used to be tightly bound to the current release, or to a particular release of say the Clarion language and Clarion templates, and you've kind of unbound it from that so the IDE can now interact with the current version or it can interact with an older version, or whatever version.

Andrew Guidroz: I can see value going backwards, but I don't see anyone buying the C7 compiler without buying the IDE.

Dave Harms: No, no, no. Yeah, why would you?

Andrew Guidroz: You got to buy both.

Bob Zaunere: Yeah, of course. That's a given.

Dave Harms: But if you want to take advantage of the changes in the IDE, it doesn't mean that you have to pay the full nickel to upgrade to… I guess we're kind of setting Bob's sales policy for him here on his pricing and stuff, but I'm assuming that it would be cheaper to upgrade to just the IDE than to upgrade to the IDE with Clarion 7, let's say, if it's called that, and Clarion.NET.

Bob Zaunere: Exactly true.

Andrew Guidroz: You have to have a certain version in mind. You're evidently thinking a lot of people stayed on 5.5, is that what you're thinking?

Bob Zaunere: Well, I know that a lot of apps have stayed in 5.0 and 5.5 because people do not want to go, they don't see the percentage in going forward to deliver a product to the customers they can't charge any more for.

Andrew Guidroz: And they're a little bit scared of the new threading model also, probably, in the back of their minds – whether or not it's…

Dave Harms: Whether or not it's a rational fear.

Andrew Guidroz: Right, right.

Bob Zaunere: There's no question that that's true, and even if they weren't scared of it, they certainly, if they – I won't say competent – but if they want to do a due diligence, then no matter what, if you move an app forward, you've got to go through the whole, entire thing and test it all. You've got to make sure your third party products are up to date. You've got a fair amount of work, whether you can do it in a week or it takes you three months – whatever it is, you have to do it.

Dave Harms: It also makes it a little bit less painful now if you've got this ability for the people who are moving ahead to C6, but maybe, or C7, they may want to have the latest, greatest version, but they still have applications, multiple applications back in previous versions. Now they don't have to deal with the clunkiness of going from a newer version

that has a feature A that I really like, having to go back and work in an older version that doesn't have that feature that I really like, because you have the feature in all of the…

Andrew Guidroz: But they would still have to own those older versions. So let's say if you had a 2003 version, your customer had a 5.5, they couldn't leave it in 2003, because they wouldn't own the 2003 version of the product.

Bob Zaunere: Well, that's true, unless that became a feature of the subscription program, which it could now that you mentioned it. Ship the older versions of Clarion with it so…

Andrew Guidroz: The fourth new product here, I got you.

Bob Zaunere: But you know, the idea of selling the IDE separately is just something we just mention casually. It has no decision been made. But I'll mention to you that there are people, a lot of people outside the U.S. in particular, where because of exchange rates, to upgrade is a very significant part of their income for the year. And so there is a possibility that we'll say, "You know what? We understand that and here's a stepping stone. It's a part way point. If you want the new features in the IDE, you'll be able to do it." But even for those people who want Clarion 7 and the new IDE all in one, there's still many people out there who have apps where they're done, they're shipping, but every once in a while they have to work on them, they have to debug something, they have to add a new features, and they're never going to be migrated forward and they don't want to migrate them forward. So I think there's some real value there to everybody.

Andrew Guidroz: What ships first, the IDE or the C7 compiler?

Bob Zaunere: The IDE is under development. The compiler isn't going through any major overhauls. So the compiler right now, I'm compiling everything with the Clarion 7 compiler myself. The IDE is compiled with the Clarion 7 compiler.

Andrew Guidroz: Does 7 imply .NET?

Bob Zaunere: No, it does not – 7 in Win32. I don't see the Win 32 world going away just because .NET's available. I do see the Win32 world continuing to evolve. There's things that the people want on the current platform, being Win32. Whether those are things to support specifically new operating systems, new versions of XP, other features of XP, Windows 2003, etc. And so, no, the Win32, I don't see a mass migration happening to Win32. I do see – I said this at DEVCON [ph?] – I do think it's prudent that people start to work with .NET soon and I do know there is a fair number of users who moving to .NET

is critical to them right now. They need it now – they have external forces, whether it be corporate or it be their customers, who have dictated, "We want .NET."

Andrew Guidroz: So what's first? What will we see from you first? Will it be .NET, C7, or the IDE, and/or what?

Bob Zaunere: The answer's yes.

Andrew Guidroz: All three? Basically you're going to try to make them peak at the same time?

Bob Zaunere: No. And definitely there's no plan to try to coordinate the schedules. They're separate projects running on their own, with their own teams and their own timeline. What I mentioned to Dave is that in the first incarnation of the soon to be born subscription program, there will be an early version of the IDE and there will be an as of version of the .NET compiler, if you buy in at that level of the subscription program.

Andrew Guidroz: How soon do you anticipate that happening?

Bob Zaunere: I'm hoping that – the original goal was the end of February, and it's looking highly probable that that will be pretty close. So within a couple weeks.

Andrew Guidroz: Do we have any pricing?

Bob Zaunere: If you're in the Clarion world, what does pricing matter? We're going to deliver you what you need, you're going to spend what you have to. Right?

Andrew Guidroz: Certainly, but I have to set my budget. You know what I mean?

Bob Zaunere: Yeah, it's not going to be anything outrageous. It's going to be fairly priced and there'll be various levels available. So you pick your level and you get in it.

Andrew Guidroz: Certainly. You know I'm in the enterprise.

Bob Zaunere: And it won't be the only option. If people aren't interested in it, they can just buy their normal upgrade or new product and say, "That's all I need." But for a lot of people, the subscription program really wasn't something that we invented. It was something that people asked us to do. And what it really was intended for is people who say, "You know, I really want to get everything you do. I'm not sure I'm going to use it all

and I don't really want to spend full price to get it all, just to find out. But if I had it, I might find a reason to use it and I'd like to have it available so that if I ever need it, I don't have to call you and wait for it, it's just there and I just pull it off the CD or download it."

Andrew Guidroz: Plus you can flatten the cost of doing business that way. Get rid of those peaks and valleys.

Bob Zaunere: Yeah. But that is a driving factor behind, absolutely.

Dave Harms: I just want to go back a second to this whole idea of one IDE and multiple versions of Clarion. Does that in fact mean that if you have the new IDE and you have Clarion.NET and you have say Clarion 7, that you'll be able to just hit a checkbox and generate an application for either Win32 or .NET?

Bob Zaunere: You know, quite honestly, that's going to be… That's not something that's entirely in our control and the reason being, okay, sure, we could say you could do that if you follow certain guidelines. For instance, unless we're going to go through and pars all your code and find all your Win API calls - this is one small example – then that code's not moving forward into .NET.

Dave Harms: Well, I guess it could move forward as unmanaged code, right?

Bob Zaunere: Well yeah, sure, if we went through and we parsed all your code and we wrapped it in the correct calls for the inter option. But we don't have any plans to be modifying people's code. They usually don't like it when we try that. So I don't think we'll do that. And the reverse is true too. If you go into your .NET app and you go, "Wow, let me look at this class library. There's this really cool thing I could do right here and I'm going to call into the class library and do it." You're not going to bring that one back to Win32 either.

Dave Harms: Let me phrase that a little differently. Could someone starting with a brand new app in the next version of Clarion create an app that could be used for both if they took the care to say put in the switches to generate conditional A, some API calls, or some .NET library calls? Or to put it in the simplest terms, could they straight generate an app, be generated for one or the other?

Bob Zaunere: The last thing you said is most likely true, although until I get some feedback from people that that's actually important and not just a, "Gee whiz, yeah, that would be cool, but I'll never really do it. I don't think I'll put much effort into it." In

theory, a straight generated app will definitely work – it probably goes beyond theory. But it isn't a design goal for either product because although it was brought up at DEVCON by one or two people, it wasn't mentioned to me by more than those one or two people, and in many cases, it probably is technically not that easy to do. I don't know, if I find out it's important…

Dave Harms: Yeah. So in fact, in a way it's an analogue to the Clarion 2 to Clarion 4 jump, right? The legacy to ABC jump. Because…

Bob Zaunere: Maybe. See, the fact of the matter is mostly everything you do in a generated app will compile and run on .NET, but I guess I'm looking at it from a different point of view. How often in the world will you say, "You know what? Darn it, I need a Win32 version of this and I need a .NET version."? I mean, how often? And then it's obviously puts the burden upon the developer to go, "So, I got this, which is great Clarion data, but now I'm going to go test it under .NET and I'm going to have to test it under Win32." So I don't know, that's a gray area for me.

Dave Harms: What you may have is you may have an application that you say, "You know what? I am now ready to migrate this application to Clarion.NET and starting off with something – or port is maybe a better term than migrate – because you could have multiple applications open at the same time. This could also make it, the new IDE would make it a lot easier to create, for instance, a Clarion.NET version of your Win32 app because it could have both applications open at the same time <overlapping talk>.

Andrew Guidroz: Did I drowse off and miss the part about multiple applications open at the same time?

Bob Zaunere: You just hadn't tuned in to the station by that time. Yeah, that's going to be possible.

Dave Harms: Not for – I guess what Bob said before was not for Clarion 6 and earlier, but for, well Clarion 7 or whatever it's going to be called, and for Clarion.NET apps you'd be able to have multiple apps open at the same time. As long as you're not working with an older set of binaries, set of ID binaries and compiler binaries.

Bob Zaunere: Yeah, let me just interject and put it this way – if you're working with let's just say Clarion 7, can you have two apps open? Yes. If you've opened up the IDE and you said, "I need to work with Clarion 5 right now, that's what I'm going to be working with this app that I need the Clarion 5 run time, I need the Clarion 5 compiler, etc.", then you won't be able to go, "Oh, let me open another window with another app and this one

will be Clarion 7 and I'm going to do that."

Dave Harms: Or another Clarion 5 app.

Bob Zaunere: Well no, you'd be able to open another Clarion 5 app presumably. That one hasn't been hurdled yet, but I believe the answer's going to be you will be able to. But the different versions of Clarion in the same instance – and here's the key – of the IDE won't be possible because of the redirection file really. It goes beyond that, but essentially what I said to Dave is when you say I'm working with Clarion 5, then everything has to be redirected to look at the templates, the lib source, the run time libraries, the drivers, etc. for that version. So you couldn't say, "Oh, now I'm going to open another window and I'm going to work with Clarion 7, because we're just not going to factor that in. But having said that, the IDE is being designed so you could just start a new instance of it, and in that separate process, you could go ahead and open up your Clarion.NET or your Clarion 7 app and be happy about it. Now, what you just said though about Clarion – there's a port, there's a migration, and then there's something in between. If we talk about Clarion 7, just say that, and Clarion.NET – can I take an app and go, "Hey, I built this app. Okay, generate for .NET. Okay, generate for Clarion 7", and same single app. Well, unless I find there's a big demand for that, we're not going to put a lot of resources into that.

Dave Harms: But if I've got a Clarion 6 app, I migrate it to Clarion 7, and then I want to migrate it to Clarion.NET…

Bob Zaunere: That should be seamless.

Dave Harms: And then, but obviously still would have to make my own changes to my embed code to convert it from any Win32 stuff I was doing, to instead using .NET stuff.

Bob Zaunere: Right, you will. We're making some allowances for that, Dave, in terms of we've at least designed…

Dave Harms: Like the string stuff you posted on the website, the _____ and so forth?

Bob Zaunere: Yeah, and there's opportunities and we may or may not go down the path of – maybe we will in the subscription program – things that you could insert a template that would go, "Okay, wrap this stuff because it's unsafe, and I don't really care how it's done. I want this block of code wrapped because I want to make these calls period", and not have to learn how to do the inter op and how to work in that world. So that's a possibility, but a straight app should migrate 100 percent. Let me just make that 100

percent clear – forget about the Win32 stuff, but the goal to do the .NET project is to say, "You migrate." And we had this conversation earlier too…

Dave Harms: Even if it's all Clarion source and no API calls, it should just be able to go straight across.

Bob Zaunere: Should be able to. And we had this – I'll repeat one time, I probably don't have to, but I want to say it to Andrew anyways – if you look in the .NET world, you can't migrate VB6 code, not without huge headaches and huge pain. And mostly it's not even recommended to try to do so, by the same company Microsoft hired to build the migration tool. They flat out have it on their information, "You really shouldn't do this unless you really need to. And if you do need to do it, expect that most of it's not going to migrate anyways, but hey, we'll migrate some of it for you." <overlapping talk> Yeah. But it just doesn't. It wasn't a goal for them – not in the cards, or just wasn't possible to redesign the language. C++ code, you're not going to have a happy time migrating that code either, but typically you don't really want to migrate code that's written at that level. Migrate – I don't even know, I'll have to think about the term migrate, because yeah, there's some issues to migrate, but right now – and the list may grow – but right now it's miniscule. Migrate can be done at the flip of a switch really. You will be in the .NET world, you will like it. You will enjoy playing with <overlapping talk>.

Dave Harms: There is no data, there is only .NET.

Bob Zaunere: No, it should be interesting. But the new IDE is really looking great and Clarion 7 will just continue down the path of <overlapping talk>.

Dave Harms: So Clarion 7 will still be called Clarion 7, it's just the IDE is needs to have some sort of a new name?

Bob Zaunere: No, no, no. It wasn't anything like that. It's not even an active thought. It's just the fact that it came up in conversation that the design of the IDE would allow someone, really who – for whatever reason, couldn't afford or wasn't ready to, or what have you, to work with something other than Clarion 7 – whether that be Clarion 6, or Clarion 5, or Clarion.NET, or VB.NET or C Sharp in not too far away future. So it may never come about, it's just a possibility that was never seriously considered…

Dave Harms: You mean I gotta take that announcement down off the website now?

Bob Zaunere: Yeah, you better yank it.

Dave Harms: All right. Okay, it's going.

Andrew Guidroz: I like the price Dave posted.

Bob Zaunere: You'll buy in at that price?

Dave Harms: That free open source IDE? That one?

Andrew Guidroz: Yeah, that's the one I was looking at.

Bob Zaunere: Yeah, the new IDE is really looking really nice. And we're eager to get it into people's hands so we start to get some feedback and we tune things and fine tune things. We'll go through the normal critique of the icons we chose – that's always a big issue.

Andrew Guidroz: Yes. Just remember that you can plan everything that you want. You can make it look beautiful, but once it goes out the door, the world will see it someway that you never imagined. You never envisions.

Bob Zaunere: Well, you know what?

Dave Harms: Like the Clarion 2 foot, yeah.

Bob Zaunere: At DEVCON, I said we're going to really allow a lot for personalization of the IDE. One of the things I thought about is a line for personalization of the icons used in the toolbar…<fades out>

Dave Harms: That's probably a good a place as any to end part one of our interview with Bob Zaunere. He really had a lot to say about the product, I thought.

Andrew Guidroz: This, to me, would make using this tool so unusual. I mean, how many times do you say, "Hey, I'm wondering where the tool is going today. Let's get on the phone with the president of the company and see what he thinks."? Should we switch to our interview with Gates next?

Dave Harms: I don't know. I think Bill had a lot of stuff to say that was really off the record. So I think we'd be in trouble if we aired that one.

Andrew Guidroz: But I think it's cool and I think the nice part is obviously Bob is

excited about the technology. He wouldn't be talking about it just to talk about it. These guys, they're not famous for going out and saying, "Let's have a marketing blitz with our president on the front page of every newspaper in the country." I mean, that's not the Soft Velocity way. But for him to want to talk about it and be so forthcoming, obviously he's seen the technology, he's playing with the technology, and he thinks it's cool.

Dave Harms: Yeah. That was my biggest impression actually from DEVCON was when Z got up there and he was giving his presentation on .NET and on the next version of Clarion, and he just looked completely comfortable. I mean, he was really having a good time. And I think some of the other stuff, where it was still maybe talking a bit more about product that really they'd inherited from the old company, I didn't sense quite that same…

Andrew Guidroz: He who shall not be named – yes, I understand.

Dave Harms: Right. I didn't sense quite the same level of maybe comfort or excitement. I think what he's obviously done is worked very hard to bring that product up to his own personal standards of what Clarion should be. But I've always felt that he didn't, that's not really something… Even Clarion 6 in many ways is not something that really has the stamp of Soft Velocity on it, and with the next version, I think we're finally seeing something that really has Bob Zaunere's vision.

Andrew Guidroz: They've taken care of all the housekeeping stuff and now they're moving forward. It probably would have been better had I actually been here to hear the first half, which I still haven't gotten to hear yet.

Dave Harms: No, no. No, no, no. That stuff stays in the vault.

Andrew Guidroz: Oh, man.

Dave Harms: You want to hear that? No, there wasn't anything interesting there. No, it was just boring stuff.

Andrew Guidroz: There you go.

Dave Harms: Maybe next podcast, I don't know. We'll see.

Andrew Guidroz: Oh, cool. Cool.

Dave Harms: And that about wraps it up for this edition of Plant Clarion. Just a reminder

for anybody who missed the first few minutes of the podcast – I don't know how you do that, but I guess you could have skipped ahead or something – that there is, Clarion Magazine is publishing e-books.

Andrew Guidroz: Woo hoo.

Dave Harms: Yeah. They're coming in March and if you subscribe or renew by March the fifth, I believe it is, you'll get an electronic coupon for two free e-books – an e-coupon for e-books.

Andrew Guidroz: Wow, an e-coupon. What a deal. Coupon, not coupon. Man.

Dave Harms: A coupon…

Andrew Guidroz: Is it progress or progress?

Dave Harms: Do you have any Grey Coupon?

Andrew Guidroz: Y'all sure talk funny up there.

Dave Harms: Not as funny as y'all talk down there. And then next podcast we'll have some of the – we'll have the secret tapes.

Andrew Guidroz: Secret tapes?

Dave Harms: We'll have the secret tapes of what Bob Zaunere really said.

Andrew Guidroz: Bob Z, unplugged.

Dave Harms: That's right. Well, thankfully if he's unplugged, it's going to be really hard to get it on the podcast.

Andrew Guidroz: Yeah, I guess so.

Dave Harms: Well, that about wraps it up. Did I say that already?

Andrew Guidroz: I think you did. I think you've wrapped it up twice.

Dave Harms: Well, we like to have these things wrapped up really nicely.

Andrew Guidroz: Put a bow on it, Dave.

Dave Harms: Put a bow on it, put a fork in it.

Andrew Guidroz: There's a bow on this – we're out of here. By everybody, see y'all next week.

Dave Harms: Okay, bye.

Reader Comments

Add a comment

- » Hey Dave - what's with all the waffle at the top... I mean...

Clarion Magazine

Compiling C with the Clarion IDE, Part 1: It's Easier Than You Think

by Carl Barnes

Published 2005-03-17

I'm not sure how long Clarion for Windows has shipped with a C compiler. A little digging showed that the C*x*CPP.DLL in the Clarion BIN directory is the C/C++ compiler, and that it existed in Clarion 4 but not 2.0. I have often wondered how I could make use of this tool. In this article I'll show you how easy it can be to include C code in your Clarion application, using this compiler. I'll also give some examples, using commonly-available C source code.

Clarion does not include just any old C compiler. It comes with the TopSpeed C++ optimizing compiler. This is the C compiler Dirty Harry would use, "cause it could blow your head clean off...punk." This is also the compiler that compiles much of the Clarion IDE (a lot of it is written in Clarion). The TopSpeed system was designed for mixed-language compiling and linking, so combining C and Clarion should be optimal.

I am not a C programmer. I can read the source fairly well, and I could probably modify working code…a little. I think it would be stupid for me to try to write classic Windows GUI programs using the TopSpeed C compiler. Soft Velocity doesn't provide the tools or files needed, plus the Clarion language, IDE and templates are better suited to the task. But there are times when C code is the best solution to a particular problem.

I was out surfing the Internet when I ran into Christophe Devine's website and his crypto page containing C source for some well known cryptography ciphers and hash algorithms. My first thought was that I could convert this C code to Clarion code. The RC4 algorithm looked easy enough, but the others involved a lot of code, and most of it doing bit manipulation. C uses what I think is nasty syntax with few reserved words and lots of special characters. This particular code also made use of compiler preprocessor macros (the # commands), which cannot be done in Clarion. It would be tedious to convert and easy to screw up. I would need to test it carefully to be sure I had converted it correctly.

Then it occurred to me to try to compile the C code directly in the Clarion IDE. If that worked it would open up a world of free open source C available on the Internet. This could be a nice contribution to Jim Kane's Lazy Programmers Society.

It took about 30 minutes of mistakes, and I had the website's RC4 algorithm compiling and working. About half of that time was trying to figure out how to use the RC4 functions and prove they worked. Getting the C code to compile was very simple! And once I knew how to do it with RC4, the AES code took less than 10 minutes. This code compiled with just a one common and tiny modification to the header file. One other bonus was the code ran at the speed of C, so it should be a little faster than my Clarion code.

Before I get to the details there are a few legal issues I probably should mention. I am not a lawyer – I am just trying to mention a few things to keep you and me out of any trouble. My intention here is not to hack, steal, hide or encrypt anything that would violate anyone's rights or US law. I just want to show you how to compile C in your Clarion IDE. The C encryption code I'm describing is copyrighted by Christophe Devine, but he releases it as free under the GNU General Public License. You will have to read the license for yourself to determine the consequences of using GPL'd code in your applications. RC4 is an algorithm of RSA Security. As far as I know they do not claim the algorithm as a trade secret and allow it to be used without license from them. But RSA does own the name RC4 so be careful how you use it. (A more generic name is ArcFour.) That may apply to other names mentioned here as well.

The encryption, hashing and other algorithms may be subject to United States Government law and restrictions, especially if you are using them outside the US. If you decide to use this code in a commercial product, or outside the US, be sure you know what the rules are, because I don't. If the NSA pays you a visit don't complain to me.

Compiling C in Clarion

C code files generally come in two parts: a .H header file with declarations, and a .C file with the code. This is identical to the way Clarion ABC classes are (well, can be) built in two parts: an .Inc file, the declarions, and a .Clw file containing the code or definitions. Here's the header file for the RC4 code:

```
rc4.h
#ifndef _RC4_H
#define _RC4_H
struct rc4_state
{
    int x, y, m[256];
};
void rc4_setup(struct rc4_state *s, unsigned char *key,  int length);
```

```
void rc4_crypt(struct rc4_state *s, unsigned char *data, int length);
#endif /* rc4.h */
```

And here's a portion of the source file:

```
rc4.c
/*  An implementation of the RC4 algorithm. Copyright (C) 2001-2003
 *  Christophe Devine. This program is free software; you can
 *  redistribute it and/or modify it under the terms of the
 *  GNU General Public License .
 ** snip **
 */
#include "rc4.h"
void rc4_setup(struct rc4_state *s, unsigned char *key, int length)
{
    ** snip **
}
void rc4_crypt(struct rc4_state *s, unsigned char *data, int length)
{
** snip **
}
```

To compile this code in your Clarion project, simply add the RC4.C file to the project. That's all there is to it. I nearly fell out of my chair when it compiled without error. So the first step is to add the .C file to a project and see that it will compile.

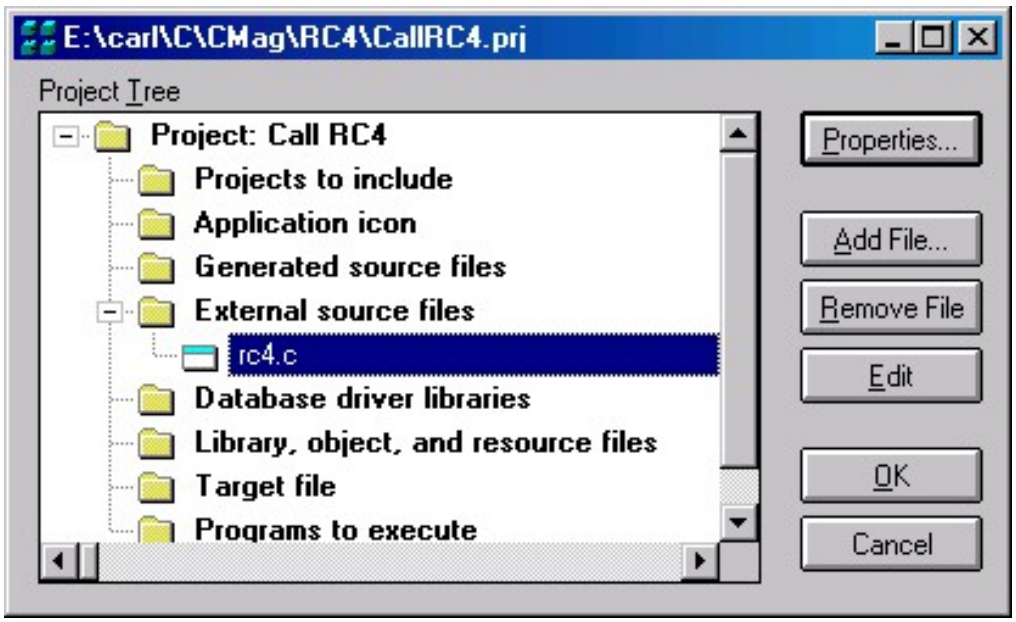

Figure 1. Adding C source to a Clarion project

While the C file throws no syntax errors, it will be compiled as C++ code and the function names will be mangled using C++ rules, meaning the compiled prototypes will include parameter

information, so that you can have procedures by the same name, but with different parameter lists, and the linker will be able to tell them apart. This will, however, cause some difficulties in the next section, so step two is to make it easier to prototype by using C naming rules (you won't be able to overload function names, but that isn't an issue here). You'll find more information on mangling in a later article section.

Wrap the prototypes with `Extern "C" {}` in the .H file as shown below:

```
extern "C" {  // Carl added, don't type Extern
void rc4_setup(struct rc4_state … length);
void rc4_crypt(struct rc4_state … length);
          }  // Carl added
```

Doing this makes the external names simply the function name with a leading underscore. For example, `rc4_setup` becomes `_rc4_setup`. You'll see how that is used in the next section. One important tip, the C language is *case sensitive*. If you type `Extern` or `EXTERN` instead of `extern` you will get a not so helpful syntax error about "expected ;:[({", and few more errors because the definition did not compile.

Prototyping C Functions in Clarion Code

To call the C functions you will need to prototype them in Clarion code, using Clarion data types and other attributes. Without a doubt this is the hardest thing required. If the data types are not obvious it may be helpful to look at the source of the function and see what it does with the data.

The main two things you need to figure out are the equivalent Clarion data type, and if the parameter is passed by value or address. If the prototype contains an asterisk (e.g. `char *`) then it is definitely passed by address. But even without the asterisk, there are cases where the parameter is passed by address.

The Programmer's Guide contains a section on Multi-Language Programming that documents the data type conversions between C++ and Clarion. I have created a variation of this table below. To keep it short I do not show every passed by address variation. If the parameter has an asterisk it is passed by address. If it does not have an asterisk it still could be passed by address, especially if the data type is complex. I also assume you will be working in *32-bit only* (in C the `INT` type is a short integer when compiled 16-bit). I am fairly certain these types also apply to 64-bit. Remember these are C base types and not Windows types.

C Type	Clarion Type

* &	pass by address – <omittable> pass by address - required
[#]	Array, use DIM(#). The name of an array is actually a pointer to the first element of the array. So arrays tend to be passed by address even though you do not see an asterisk.
char, unsigned char, signed char	BYTE, but probably a string type
char *, char[]	CSTRING or STRING (need RAW). If the array has a dimension number it tends to be a fixed length string e.g. char Digest[16] is a STRING(16). A char * could be a *BYTE, but it is rare.
struct	GROUP (need RAW)
unsigned short	USHORT
signed short, short	SHORT
int, signed int. long, signed long, signed	SIGNED or LONG UNSIGNED is Equate(Long)
unsigned, unsigned long, unsigned int	ULONG – these have disadvantages, see below.
float	SREAL
double	REAL
void	nothing to enter
Void *	This is a pointer to something. Use a LONG or UNSIGNED.

The hardest part of this conversion is usually figuring out what data is a STRING or CSTRING Clarion type. The C language does not have a string type per se; instead, it uses arrays of characters normally terminated by a zero byte. If a char type is passed by address (using an asterisk e.g. char *) then it probably is a CSTRING, but could be a byte. If you are passing a CSTRING, STRING or GROUP you will almost always need RAW on the prototype; otherwise, Clarion also passes the length of the string; for this to work the C function must be prototyped to receive the length, and that won't often be the case.

One other tip is to avoid using the ULONG type in Clarion. The object code created for ULONG math uses the decimal library, which is much slower than the code used for a LONG. The UNSIGNED type is equated to a LONG, which is preferable. For most of my code I tend to use SIGNED, which is also equated to LONG in 32 bit code. Don't worry about the use of unsigned longs in C, as these are

identical to a signed long.

So now that you have the prerequisite type conversion knowledge, let's get to the next step of creating the Clarion declaration of the C functions. I think the best way to do this is to take the .H file and convert every line of it to a Clarion .Inc file. I can include that file in my project and have everything required to call the C functions.

I typically start by copying the .H file and renaming it XXX_H_Inc.Inc. One thing I found caused problems was a double extension like xxx.h.inc, so don't do it. A second and bigger problem is the files you download may be Unix files with a line termination of <10> instead of <13,10>. When you try to compile them in the Clarion compiler they will show syntax errors for perfectly good code. The typical syntax error will be an illegal character at the end of the line where there is not bad character…you can see. I fixed these files by loading them into UltraEdit and doing a Unix to DOS conversion. (Editor's note: You can open the file in Wordpad and save it – Wordpad will add the Windows style CRLF.)

My RC4.H to Clarion converted file is shown below:

```
RC4_H_Inc.Inc
RC4_State_Type Group,TYPE       !struct rc4_state
x           Signed              !{
y           Signed              !   int x, y, m[256];
m           Signed,DIM(256)     !
              End               !}
  MAP
    MODULE('rc4.c')
      !void rc4_setup(struct rc4_state *s, unsigned char
          ! *key,  int length);
      rc4_setup(*RC4_State_Type State, *String KeyString, |
            Signed KeyLen),name('_rc4_setup'),RAW
      !void rc4_crypt(struct rc4_state *s, unsigned char
          !  *data, int length);
      rc4_crypt(*RC4_State_Type State, *String InOutData, |
            Signed DatLen),name('_rc4_crypt'),RAW
    END
  END
```

The C struct is now a Clarion GROUP, and it contained some INT types that are a Clarion SIGNED or LONG variables. I added the TYPE attribute or the group would have been allocated memory.

It is fairly typical to have an set of functions that require you to pass a group as a parameter, so the code can use the group (struct) to maintain state information and other data. (In OOP this practice has been formalized, and the data is wrapped in the Class structure.) Unless your Clarion code needs to access the fields within the structure you do not have to worry about getting the types exactly right, as long as you have the length of the group correct. This is sometimes called treating

the data as *opaque*. I could have defined a single field in the group as `STRING(4 + 4 + 4*256)` and it would have worked as well.

Clarion prototypes must be contained in a `MAP` and `MODULE` to compile correctly, so I wrapped them accordingly. A program can have more than one `MAP` so including this file will not pose any problem. The data types were converted as shown. The `RAW` attribute was added so the compiler would not pass length information along with `STRING` and `GROUP` parameters. The name attribute uses the C convention of preceding the name with an underscore. If you do not want to use `extern "C"` you'll need the C++ mangled name.

A common item you may see is a #define which is the same as a Clarion `Equate`. If the equates are useful in calling the C functions, then they should be converted and put in the Clarion include file. Many times a `#define` will be used to create a shorter name for a data type, e.g. `#define uint32 unsigned long int`. I would not convert that to an equate; instead I would simply use a Clarion `UNSIGNED` anywhere I found a `uint32`.

Another use for a `#define` is to only include the .H file once. This can be seen at the top of the file with two lines: `#ifndef _RC4_H` and `#define _RC4_H`. This can be done in Clarion with an `OMIT` and `EQUATE` as you'll see in almost every ABC Inc file, or an `Include` with the `ONCE` attribute. For this simple example I did not do it but it's probably a good idea.

Including C Modules into the Compile Stream

Above I said that you needed to include RC4.C in your project as a source module so it will be compiled. The `MODULE('RC4.C')` line in the `MAP` does not get the source compiled, so failure to include the source file in the project will result in linker errors. This would need to be done in every project where you want to use the C source, making it a small pothole waiting for you to trip up and spend 30 minutes trying to remember how this all works. You can avoid this step by adding the C modules to the project compile list automatically, using this Clarion trick in the include file:

```
CompileRC4   CLASS,TYPE,LINK('RC4.C',_ABCLinkMode_), |
                        DLL(_ABCDllMode_)
             END
```

This empty class serves no purpose except to tell the project system to compile and link the RC4.C source file.

In C6 this can also be done by adding either of the following pragma lines to the include file:

```
PRAGMA('project(#compile rc4.c)')  !option 1
PRAGMA ('compile (rc4.c)')         !option 2
```

Next week I'll show how to call the RC4 functions from Clarion.

Download the source

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

Add a comment

- » Finally... I have been waiting for a good article...

Clarion Magazine

Progress Bars for Queues: Using The ABC QueueProcessManagerClass

by Bjarne Havnen

Published 2005-03-18

Recently I found myself in a situation where I once again needed to process a queue. Usually this is trivial, but in this particular situation I had to add a little pause between each record processed, and then of course display the progress to the user

Until now, I have always just made a regular window with a timer and processed each record in the timer event, as follows:

```
SomeProcedure Procedure
CTR Long
  Code
...
ThisWindow.TakeEvent  Procedure
  Case Event()
    Of Event:Timer
          Ctr+=1
        Get(MyQueue,Ctr)
        If Errorcode()
           Window{Prop:Timer}=0
          Post(Event:CloseWindow)
        Else
          Do ProcessQueueEntry
          Do CalculateProgess
        End!if
```

I know this isn't much code, but add to that the handling of a Cancel button, and sometimes record validation, and it starts to get messy. In addition I will often, voluntarily or not, end up with a lot of different progress window designs. On top of that, I am not very fond of the separation of code into routines. It doesn't take many routines to lose track. A class is a much better solution when I need to detect an error or apply a change.

But what code would I need, in order to achieve a reusable solution for processing queues using a standard

window? Happily, I found an undocumented Clarion class that, with minor extra code, does the job nicely. In this article I will describe that class, and in Part 2 I'll add a template to make using that class even easier.

The Lazy Programmer's Society

After a few years of programming in Clarion, I have become a very lazy programmer. Writing the same code and designing a new window every single time I need some functionality is way too much to ask. I am familiar with classes and templates, so I thought about making a class and template wrapper to write this queue processing code for me.

This code would:

- Process each record in a queue, either the queue is passed as parameter or not
- Display the progress to the user
- Warn when there were no records
- Terminate the procedure when finished.

Does this specification look familiar? Yes, it is just the same as any Clarion process does today.

Sometimes when I have a bright idea, I find out later that someone already had the same idea and did something about it. That is possibly one of the reasons I am not a billionaire. With this in mind, and knowing that Clarion 6 introduced a new option in reports, namely the ability to use a queue as data source, I knew there should be something already available.

I opened up my ClassViewer and pressed "Q". There, right before me was a `QProcessManagerClass`! A little examination showed that this class didn't solve my problem, but a derived class, the `QueueProcessManagerClass` (QPMC) had all the functionality I needed, right out of the box, but not yet documented. According to Bob Foreman, SoftVelocity's head of documentation, this class is used in some ADO processing, with which I am not familiar.

On finding this class, I was surprised to learn that it is not in use in any regular process. Even the reports where you can select Queue as a data source don't use it, they just add a lot of code to override the default report behaviour. Very strange, I thought, given the design of this class.

Derive or not?

In many cases, when it is necessary to add some extra functionality to the ABC templates, all that is required is to derive a class and tell the templates to use the new class instead. My attempt to just specify the QPMC on the classes tab of a Process procedure resulted in seven compile errors, because the QPMC is not based on the `ProcessClass`. Of course, given that the `ProcessClass` is based on the `ViewManager` and the QPMC doesn't need a view at all, the separation of the two classes made perfect sense.

My first thought was to derive the `ProcessClass` instead and replace the necessary methods with my own, but just the fact that this would still require me to set the primary file for a process felt wrong. So

instead of using the process template, I decided to use the class directly.

Using the QPMC

So, what is required use the QPMC to process a queue? Actually, it only takes six lines of code to achieve my primary goal, but if I increase the code a little, I will have a more dynamic approach.

In the example, I have a global queue. It is populated in the first procedure, and I need to process this queue frequently.

First of all, any process needs a window. To do this I create a new procedure, give it any name, and select Window – Generic window handler as the template, as in Figure 1.

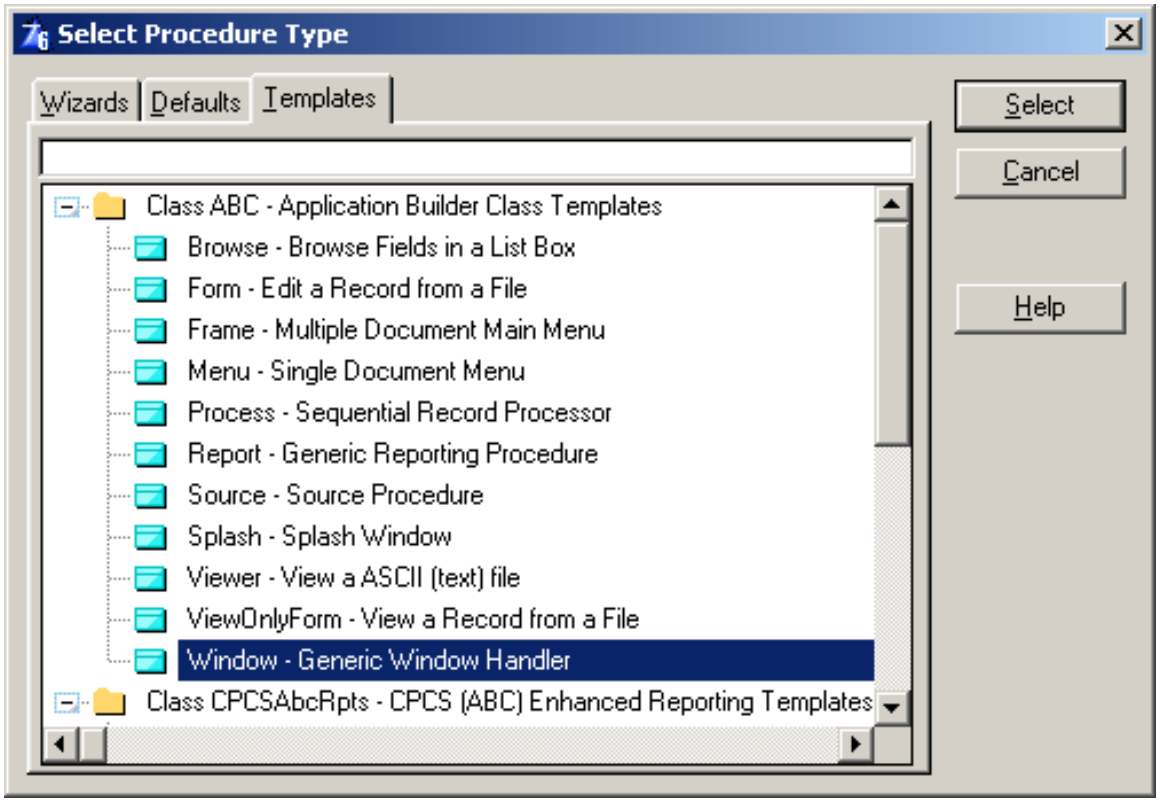

Figure 1. Selecting the procedure type

Next, I hit the Window button and select the ProgressWindow (Figure 2).

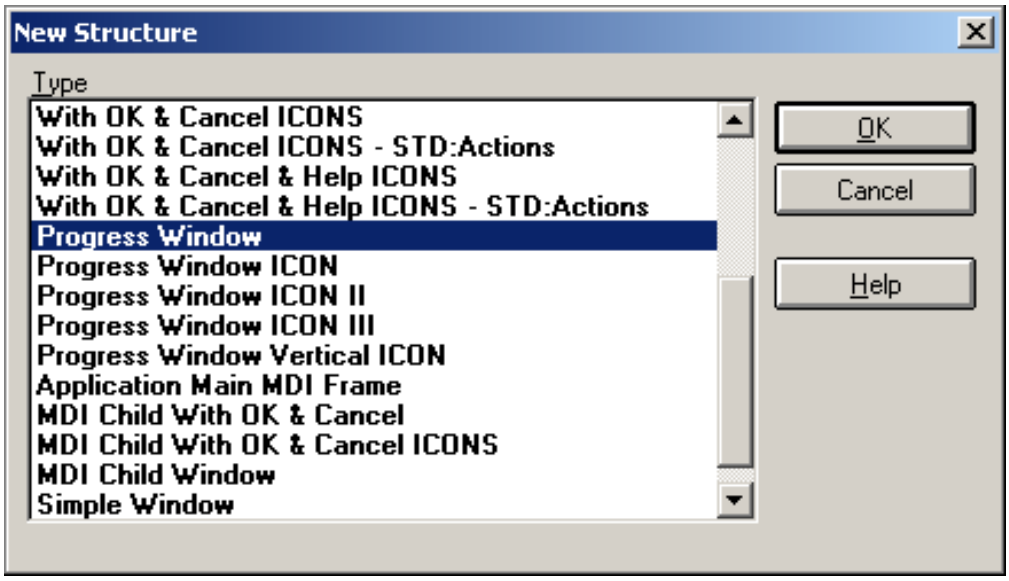

Figure 2. Selecting the progress window

Because only the Process and Report procedure templates automatically declare all the variables needed by the progress window, I need to declare a variable for the progress bar, called `Progress:Thermometer`. I insert this into the local data dialog (Figure 3).

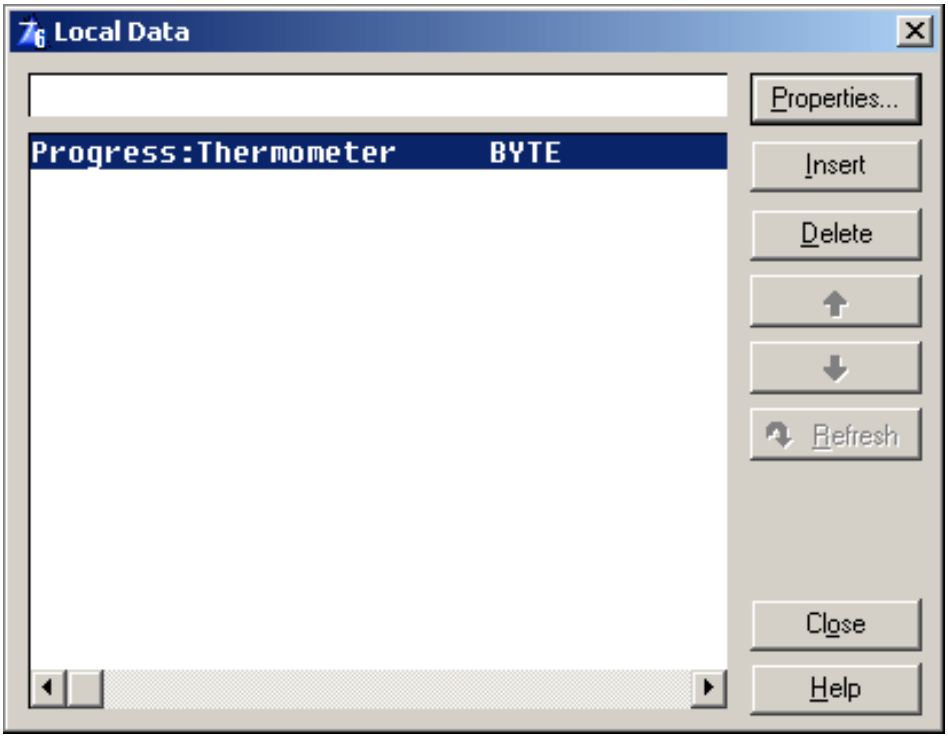

Figure 3. Adding the Progress:Thermometer variable

Now, for the code.

My globally declared queue has a very simple structure: a description field for me to see what record is processed, and a value field which I will alter in the process.

```
DisplayQ Queue,Pre(DQ)
```

```
Description     String(20)
Value         Long
 End
```

In the example I am going to perform one of two actions: either I will change all items in the queue, or I will do a display of each item in the progress window. In order to distinguish between the two actions I need to tell the procedure what I want to do. I am not very fond of the GlobalRequest approach, so I just pass a parameter. I enter the following in the prototype and parameter fields.

```
(Byte pRequest = ViewRecord)
```

The name of the parameter is pRequest in order to avoid conflict with all the class method parameters named Request. I will call this procedure this way:

```
ProcessQ(ChangeRecord)
```

If your coding practice requires you to use the `GlobalRequest`, then set the request as for any other form and test for `ThisWindow.Request` in the process. I doubt that it will ever be common to call a process in any of these ways; I am doing it merely to illustrate the point.

Building the engine

In order to use any class, the application needs to know it exists. The only place all the classes are available is in a data DLL (which is beyond the scope of this article); in any other application it is mandatory to include the header file. Normally templates do this for you, but there is no template for the QPMC (at least not until Part 2).

The QPMC is declared in Qprocess.inc. Depending on whether I need it at the module level or in the entire application, I enter the following code in the embed for Module Data or in the After global includes global embed

```
Include('qprocess.inc'), Once
```

Now the class is available to the procedures declared in that particular module or in the entire application, depending on where the header is included.

Next, I view the source of the procedure using Edit|Source, and I locate some free space in the data section

I need access to two methods of the QPMC: `TakeRecord` means do an activity for each record; `TakeNoRecords` is called when there is no data at all. If you prefer not to report to the user when nothing has been done, you can feel free to ignore any reference to the `TakeNoRecords` procedure in the following discussion.

I need a derived class declaration in order to achieve my goal. Here is what I enter in the data section of the procedure.

```
ThisProcess   CLASS(QueueProcessManagerClass)
TakeRecord     Procedure(),Virtual
TakeNoRecords     Procedure(),Virtual
 End
```

I have declared two virtual methods, which are methods the object will call instead of the base class methods with the same name. Next I do a CTRL+PGDN and find the embed called "Start of Local Procedures". This is where any class method has to be defined, and above this embed you will find all the methods of the `WindowManager` instance (which is called `ThisWindow`). I enter the definition of the two methods as follows:

```
ThisProcess.TakeRecord Procedure()
 Code
ThisProcess.TakeNoRecords Procedure()
 Code
```

After entering the above code in the local procedures embed, I can compile and run the application, but it's not doing anything meaningful.

Starting up

At this point, there is no connection between the defined window or queue and the `QueueProcessManagerClass`. If you examine the code in `ThisWindow.Init` for any process or report, you will see that the process needs to know about which controls should perform the different actions; there is no automatic behavior for aborting the mission or updating the progress bar. Also, the `QueueProcessManagerClass` instance needs to know what queue it is supposed to use.

The ABC classes are quite consistent when it comes to labelling. In anything that has to do with windows or controls you will find `Init`, `Run`, and `TakeEvent` methods. The QPMC is no exception.

In the embed following the opening of the progress window, I entered some initialisation code.

```
ThisProcess.Init(DisplayQ) !use this queue
ThisProcess.Run() !Run immediately
```

The `Init` method tells the QPMC to use my queue for processing, and the `Run` method sets the default values for different properties, like telling the process to close the window when saved (examine qprocess.clw for details). If you don't want the window to close, you can call ThisProcess.SetCloseWindow(False) afterwards. This can be handy if it is necessary to perform a timed queue process in a form or browse

In any Window, Browse, or Form procedure, events are first handled by the `WindowManager`. Many window-related ABC classes implement the `WindowComponent` interface, and these interfaces are registered with the `WindowManager`, which then automatically calls the appropriate event handling methods in those classes. Neither `ProcessManager` nor the QPMC implement this interface, so I have to

call the event handling methods manually.

In the windowmanager `TakeEvent` embed I connected the window with the process by calling the familiar `TakeEvent` method:

```
If Not ReturnValue !no other actions performed
    ReturnValue = ThisProcess.TakeEvent()
 End!if
```

I test for `ReturnValue` before calling the process. One of the reasons is that the `WindowManager` uses the `ReturnValue` to indicate if the window needs to shut down, or if the `ACCEPT` loop for some reason should `CYCLE`. I can't assign a value to such a vital variable without testing for a value already assigned.

The `TakeEvent` method handles the `Timer`, and any known buttons. It sets the timer of the window if it has been blanked, but this can only be done if an event occurs or if you call `TakeEvent` manually. The `Open()` method handles the setting of a custom timer.

What I had done so far made the QPMC process one record of the queue for each `Event:Timer`, but it did not update the progress bar, which was my primary reason for using a process in the first place. It is time to connect the progress control and `QueueProcessManagerClass`. The `SetControls` method is declared as follows:

```
QProcessManagerClass.SetControls    PROCEDURE(STRING pText,|
       SIGNED pControlProgress=0,SIGNED pControlCancel = 0,|
           SIGNED pControlText = 0,SIGNED pControlPause = 0
```

The first parameter is the "Completed" text, and the rest are the field equate labels of the controls on the window. I am not going to use the Pause button, but it could be implemented easily, although there is a bug in the `TakeEvent` method that makes the pause button only pause, not restart when already paused. I have implemented a workaround it in the example.

Alter the first initialisation code so it looks like the following:

```
ThisProcess.Init(DisplayQ)  !use this queue
ThisProcess.SetControls('Completed',?Progress:Thermometer,|
 ?Progress:Cancel,?Progress:PctText) !no pause button
ThisProcess.Run()     !RUN Immediately
```

If you want to you may adjust the speed of the process. The default is to process one record each 10/100 second. This can be adjusted with a call to `SetProgressLimits`, and the `Open` method must be called afterwards. If you don't need to adjust the speed, you can leave out the two calls

With these additions, the initialisation could look like this:

```
ThisProcess.Init(DisplayQ)  !use this queue
ThisProcess.SetControls('Completed',?Progress:Thermometer,|
                            ?Progress:Cancel,?Progress:PctText) !no pause
```

```
ThisProcess.SetProgressLimits(100,1) !one record each second
ThisProcess.Open()    !set timer
ThisProcess.Run()     !RUN Immediately
```

Compile and run the app at this stage, and you will see a progress bar finishing in seconds = Records(Q).

Actually doing some work

There's no point in a process that doesn't actually do anything, so go back to the embed LocalProcedures. Here is a sample of how you can call the TakeRecord method:

```
ThisProcess.TakeRecord Procedure()
 Code
 Case pRequest
 Of ViewRecord
    ?Progress:UserString{Prop:Text}=DQ:Description |
          & ' ' & DQ:Value
 Of ChangeRecord
    DQ:Value+=1
    Put(DisplayQ)
    ?Progress:UserString{Prop:Text}='Changed ' |
           & Dq:Description & ' to ' & DQ:Value
 End
```

And here is a possible TakeNoRecords method:

```
ThisProcess.TakeNoRecords Procedure()
 Code
 Message('No records to process','Mission aborted')
 Parent.TakeNoRecords() !code to close window
```

The TakeNoRecords method needs a parent call, because this method closes the window if no records are found. The TakeRecord method doesn't do nothing in the base class, only in the object, so a parent call is waste.

As I mentioned, I am lazy. Except for looking pretty much like ABC, and being pretty tidy if I might add, this code doesn't give me that much. After this initial attempt, I found I wanted to go further. I wanted record validation, a pause button and anything else I usually used in a process. So I wrapped everything up in a template. I'll describe that template next time.

Download the source

Reader Comments

Add a comment

Clarion Magazine

Compiling C with the Clarion IDE, Part 2: Calling C from Clarion

by Carl Barnes

Published 2005-03-22

Last week I showed how easy it is to compile C code in Clarion, and I also converted the RC4 function header file to something Clarion understands. And now that the .H file has been tweaked, and the Clarion H_Inc file created, calling the RC4 functions is simply a matter of including the Clarion H_Inc file as shown below, and then calling the functions:

```
    program
    include 'rc4_h_inc.inc'
RC4State    like(rc4_state_type)
RCKey       string('KingsPlayChessOnFridaysGenerally')
RCdata      string(256)
RCOrig      like(RCdata)
LenD         long
    code
    rc4_setup(RC4State, RCKey, size(RCKey))
    RCdata='6011-1234-1234-1234'
    RCorig=RCdata
    LenD = len(clip(RCdata))
    rc4_crypt(RC4State, RCdata, LenD)
    !MUST do the Setup again so vector is in the initial State
    rc4_setup(RC4State, RCKey, size(RCKey) )
    rc4_crypt(RC4State, RCdata, LenD )
    message('Decrypt:|' & clip(RCdata) & |
            '||Original:|' & clip(RCorig) &|
            '||' & choose(RCdata=RCorig,'Passed','Failed') )
```

To make the calling of your C code even easier you may wish to wrap it in a Clarion ABC compatible `CLASS`. This could take care of type conversions and ensure the code is called properly. In the downloadable source at the end of this article you can find my `CBRC4Class` wrapper for the RC4.C functions.

Now that you know how to compile and call the C source code, you may be wondering what it actually does.

What is RC4?

The name RC4 is short for Ron's Code version Four or Rivest Cipher Four. You can read more about it on this Wikipedia page. Ron Rivest is an MIT professor and really smart guy who invented all kinds of crypto-magic stuff. You can read more about him and some articles on his home page.

RC4 is known as a stream cipher. It is designed for reasonably secure encrypting of a stream of data very efficiently. It is commonly used in SSL. There are many details to doing security right and you are not going to get them in this article. There were some recent articles in Clarion Magazine, and you can read more about the topic by searching for cryptography.

If you look at the code for RC4 and see how it does it's encryption, you'll find it amazingly short. You call the `RC4_Setup` function and pass it a key string to initialize a state table. RC4 uses an XOR method, so the `RC4_Crypt` function does both the encryption and the decryption. The stream of data passing thru the `Crypt` function changes the XOR table so that the way the data is encrypted is always changing.

The RC4 algorithm is available on the web in a number of languages, which makes it easy to exchange encrypted data with other programs. Since you have the source you could slightly (and carefully) tweak the algorithm to make it a little different and harder to break. Another option for making it unique is to mix the real data with garbage data before encryption. It is also possible to perform RC4, and many other methods, using the Windows Crypt API as described in the ClarionMag MSCrypto article.

Compiling the AES Algorithm

I have also included an example of the AES encryption C code with the download of this article. The source code required is a lot longer than RC4. But the process to compile and call it from Clarion proved just as simple. The main challenge was figuring out how to call the functions correctly since they were undocumented. For instance, by looking at the source I discovered that the `Set_Key` functions has a bits parameter that only takes three values: 128, 192 or 256. To make this easier to remember I added some equates to the H_Inc file.

The MD5 Algorithm

Next on my list was the hashing algorithm know as MD5. This takes a block of the bytes and computes a 16-byte or 128-bit hash number that should be unique. MD5 is kind of like CRC32 on steroids. (Currently steroids are not banned in computer programming.)

Something you might do with MD5 is compute the hash value of a password, and store that hash in a database instead of the password. This way the password is not visible and there is no way to turn to hash back into password. To validate you run the entered password through MD5 and compare the hash keys. Since passwords tend to be short and from a very limited set of about 96 characters, using an MD5 hash is not secure unless you salt the value, that is you mess up the password in a repeatable way. One way to do that is encrypt it with RC4.

MD5 is no longer considered absolutely secure and SHA 1 or 2 are higher security choices with 20 and 32 byte hash results. The source for calculating those hash values is also available from the same website as RC4, and should compile using the same process. I have included a working SHA1 example in the download. As long as your application is not hiding government or banking secrets an MD5 salted password will probably be plenty secure enough.

Compiling MD5 and String.h

My first step in getting MD5 to work was just to see if the MD5.C file would compile by adding it as an external source file to a project. It threw a syntax error of "file not found string.h". I pulled up MD5.C in an editor (I like Ultraedit since it does C syntax highlighting). The relevant excerpt is below:

```
#include <string.h>
#include "md5.h"
```

I knew that string.h was a header file that contained the prototypes for the C standard library string functions. I also knew that while Clarion shipped with a C compiler it did not ship with any of the .H files needed to do most serious C programming. My first thought was I could probably have made the Microsoft or GNU C compiler .H files work for the basic functions required. It's all supposed to be ANSI standard C and they are just prototypes.

But then I remembered that a long time ago I took Clarion Corporation up on special offer to buy the complete TopSpeed DevKit. I tracked down the diskettes and all of the .H files (which I've included in the downloadable source, with SoftVelocity's permission). The files were dated 1991 so I had some doubts that they work. But I also knew the basic C libraries don't change. I put the String.H file into my directory, hit compile and got an error that I was missing the Inline.H file. I added that file and then the code compiled. Most of the MD5.H file from this webpage is shown below. All I needed to do was add the extern "C" definition.

```
#ifndef uint8
#define uint8  unsigned char
#endif
#ifndef uint32
#define uint32 unsigned long int
#endif
typedef struct {
    uint32 total[2];
    uint32 state[4];
    uint8 buffer[64];
} md5_context;
extern "C" {  //Carl added
void md5_starts( md5_context *ctx );
void md5_update( md5_context *ctx, uint8 *input, uint32 length );
void md5_finish( md5_context *ctx, uint8 digest[16] );
          }  //Carl added
```

Making the MD5.H Clarion Include File

The next step was making a Clarion Inc file from the .H file so I could call the functions from Clarion code. Since I had been bitten by Unix line termination problems, I first loaded the .H file into UltraEdit and did the Unix to DOS conversion, then saved it as MD5_H_Inc.Inc.

The first thing the .H file does is `#define` some data types simply to shorten the code. The `uint32` is a Clarion `SIGNED`. The `:uint8` is a character type and used to pass strings.

The prototype for `md5_finish()` proved perplexing. I knew that the last parameter had to be the returned hash value, but why didn't it have an asterisk on it to indicate pass-by-address? I did find a note in the JPI C Language Tutorial guide that said "the name of an array is actually a pointer to the first element of the array". So I assumed it was passed by address, and adding the `*` to the parameter worked. To ensure that I always pass the 16 byte string the function is expecting I created a `GROUP` and used that in my prototype. It's good to program defensively and make it hard to call a function the wrong way. Wrapping these functions in an ABC class could accomplish the same thing.

The types all had to be converted to Clarion. The procedure prototypes were wrapped in a `MAP` and each prototype had `RAW` and `NAME('_xxx')` added. Here's the final MD5_H_Inc.Inc file:

```
md5_context   GROUP,TYPE   !typedef struct
total   UNSIGNED,DIM(2)    !    uint32 total[2];
state   UNSIGNED,DIM(4)    !    uint32 state[4];
buffer  BYTE,DIM(64)       !    uint8 buffer[64];
             END           ! md5_context;
!md5_finish has an OUT that must be a STRING(16) or Array
! of 16 bytes so make it a GROUP to insure proper use
md5_Digest16    GROUP,TYPE
Digest              STRING(16)
DigestArray         BYTE,DIM(16),OVER(Digest)
                END
  MAP
    MODULE('md5.c')
      md5_starts(*md5_context ctx ),RAW,NAME('_md5_starts')
      md5_update(*md5_context ctx, *STRING input, |
            UNSIGNED length ),RAW,NAME('_md5_update')
      md5_finish(*md5_context ctx, *md5_Digest16 Digest16)|
            ,RAW,NAME('_md5_finish')   !uint8 digest[16] );
    END
  END
CompileC CLASS,TYPE,LINK('MD5.c',_ABCLinkMode_),DLL(_ABCDllMode_)
         END
```

Testing MD5 and String.H Redux

The below code is all that is needed to test calling the MD5 C functions:

```
    program
    include 'md5_h_inc.inc'
Ctx         like(md5_context)
Digest      like(md5_digest16)
HashData    string(1024)
    code
    HashData='abc'
    md5_starts(ctx)
    PassData = HashData
    md5_update(ctx, HashData, LEN(CLIP(HashData)))
    md5_finish(ctx, Digest)
```

My first run went BOOM and threw an access violation error. Stepping through the C code in the debugger I found it crashed when using `memcpy()`. Oh darn, I guess my String.H file would not work. But then I remembered the Inline.H file I had to add. To make a long story short the Inline.H file contained inline assembler that was crashing. If I edited the String.H file to not use the inline functions my code worked. It makes sense that 16-bit assembler op codes are not going to work compiled as 32-bit.

I could not figure out how to configure pragmas or defines to get rid of the inline code. So I just commented out some of the lines in String.H as shown below. I also removed the Inline.H file so I would be sure it was not used.

```
//Carl// #if (!((defined _INLINE) & (defined __LANG_EXT__))) || (defined __STDC__)
#ifdef __cplusplus
extern "C" {
```

```
#endif
int     memcmp(const void *_s1, const void *_s2, size_t _n);
void *  memcpy(void *_dest, const void *_source, size_t _n);
void *  memset(void *_s, int _c, size_t _n);
#ifdef __cplusplus
}
#endif
//Carl// #else
//Carl// #include <inline.h>
//Carl// #endif
```

In the article download I have included all of my .H files (with SoftVelocity's permission), but I have only tested the String.H file and a few others. To reiterate, these files are from 1991. There was no 32-bit, no Unicode and Windows was version 2 or 3. Before using a file you should give it a quick read through. For example, the Limits.H file had defined the `INT_MAX` as 32767 which is the 16-bit maximum. I fixed that file and you'll find it in the download.

If you are going to make frequent use of these H files you should put them in a common directory. I would not put them in LibSrc, I would create a new subdirectory off your Clarion folder named CLibSrc. Next add a line to your Clarion RED file like this: `*.H = ;%ROOT%\ CLibSrc; %ROOT%\libsrc` so the compiler will find the files. Also add the same line for `*.C` and `*.CPP` to allow putting common C modules there and using them in multiple projects.

Next week I'll talk a bit more about naming conventions and name mangling.

Download the source

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

Add a comment

Clarion Magazine

Compiling C with the Clarion IDE, Part 3: Naming and Mangling

by Carl Barnes

Published 2005-03-28

In Part 1 of this series I gave an introduction to C naming conventions. The Clarion CPP compiler offers other extern naming conventions, including Pascal, Modula2, C++ and the already covered C. The Pascal and Modula2 *naming* conventions are identical and should not be confused with the Pascal *calling* convention. I like to use Modula2 to avoid this confusion. Modula2 differs from C in that no underscore is prefixed, and the name is uppercase (which is not important since the linker is not case sensitive). You can add your own prefix to the external name by putting a period-prefix after Modula2 like this: extern "Module2.Crypto". The external name will be *`Prefix`*`$`*`FuncName`*, e.g. `Crypto$RC4_setup`. If you were using a lot of different C modules this option could avoid name conflicts.

The C++ extern method forces standard C++ name mangling and ignores any pragmas in effect. You get this by default if no extern is defined. The reason I mention it is you may want to mix methods in the same file and `extern "C++"` allows you to switch back to mangling from C or Modula2. You can put an explicit extern first on each prototype line as shown in the code below:

```
extern "Pascal" int funcP(int i);  // FuncP
extern "Modula2" int funcM(int i);  // FuncM
extern "Modula2.Carl" int funcMCB(int i);  //Carl$FuncMCB
extern "C++" int funcCPP(int i);  // _funcCPP@Fi
extern "C" int funcC(int i);  // _funcC
```

To Mangle or Not to Mangle?

What's in an extern name? Would a `Rose()` function mangled as `_rose@F` not smell as sweet? I think the best move is to not use any special external naming and use the C++ mangled function names. In the below two code snippets I show a C prototype and Clarion code with the `NAME()` using the C++ mangled name. The extern "C++" is not really required.

```
extern "C++" void easter_date( const long year, int *month, int *day);

easter_date(long year, *signed month, *signed day),NAME('_easter_date@FlPiPi')
```

While many think that the only benefit of mangling is to allow overloading, it also offers an important form of *type checking*. Let's say you implement the above code and get it working perfectly. You like it so much you put your code in LibSrc for reuse. Then someone else changes the C code prototype so month and day are a `*char` because the maximum return value is 31. That will change the mangled name, and your code will no longer link. This is a

good thing because you may need to adjust your Clarion prototype. Had you used `extern "C"` you would not get any error and your code might simply crash.

So how do you get the C++ mangled name to use in the Clarion `NAME()` attribute? Once your C code is compiled, open the .Obj file (it's in the \obj32 directory) in an editor. There will be a lot of binary junk, but just search for `funcname@F`. After the `@F` you'll find the mangling ready to copy/paste. Don't forget the leading underscore, it's easy to miss with all the special characters.

The mangled name also appears in the MAP file, but there is a Catch-22. If the function is not called the linker will "smart link" it out, but you can't call it until you know the mangled name. You can turn off smart linking by adding `#pragma module(smart_link=>off)` to your C source module. If you had a lot of functions to define this would be preferable to digging through object code.

Clarion and C++ Mangle the Same

Clarion actually uses the C++ naming conventions so it is possible to not have any Clarion `NAME()`. All you have to do is turn off the leading underscore (I'll show how in a moment) and get the data types exactly the same. This work pays off by giving you overloading and type checking on both sides. That is, if you have the Clarion types or the C type wrong, the compiler/linker will give you an error. The alternative result of a wrong data type could be a runtime crash to debug, or an incorrect result.

Until you get this skill perfected it can be a trial and error affair. By turning off smart linking with `#pragma module(smart_link=>off)` or looking in the object file you can find the name assigned by C++. The Clarion linker error message will tell you the name as Clarion mangled your prototype. The Pro2Exp example (in the Clarion examples directory) will take a Clarion prototype and show you the C prototype. I have found it to be quite useful. It has at least one bug; it shows a Clarion Byte as a `byte` in C, when that should be an `unsigned char`. It also does not handle Clarion `Signed` or `Unsigned` types, which are both a C `long`.

The C++ default extern name has the leading underscore. To match Clarion this must be removed by adding the pragma as shown below. The Save/Restore insure that no other prototypes are affected. Below the C code I have show the Clarion code.

```
#pragma save
#pragma name(prefix=>"")  //so no leading underscore
void easter_date(const long year, short *month, short *day);
#pragma restore
easter_date( signed year, <*short month>, <*short day>)
```

The difficult requirement is you must get the Clarion and C++ data types *exactly* the same from a mangling point of view. This has a small bonus in knowing you have your types exactly right.

C code tends to use the `INT` type frequently. An `INT` is a short in 16-bit; otherwise, it's a `long`. This mangles to an `i`. Clarion does not have this mangling type, so you must change the C code to use a `Short` or `Long`, which mangle to an `s` or `l`. Rather than change all of the code, you can use the C preprocessor commands and safely redefine the `INT` type to a long just once:

```
#undef int
#define long int
```

A similar problem exists for the C `char` type that mangles as a `c`. The Clarion `BYTE` mangles as a `Uc` so the C code needs to be changed to an `unsigned char` to match.

There are also a few things to know to get string passing exactly right. There are further complications, like parameters passed by address must be prototyped in Clarion as omittable. As I noted above, Pro2Exp can help you get these right. I also included a text file with a table in the mangle directory of the downloadable source.

While it can be some work I think the ideal situation is getting the mangling to match. But be careful changing the data types in the C code, or you may change the way that code works and break it.

The Easter Example

Easter is going to be March 27 in 2005. I know that because I downloaded a C function library from the Project Pluto website. This function is simple enough to convert to Clarion code, but I learned a few things trying to implement it as C code. Here's the original code provided:

```
void easter_date( const long year, int *month, int *day)
{
   const long year2 = year + 5700000L;
   const long a = year2 % 19L, b = year2 / 100L, c = year2 % 100L;
   const long d = b / 4L, e = b % 4L, f = (b + 8L) / 25L;
   const long g = (b - f + 1L) / 3L, h = (19L * a + b - d - g + 15L) % 30L;
   const long i = c / 4L, k = c % 4L, l = (32L + e + e + i + i - h - k) % 7L;
   const long m = (a + 11L * h + 22L * l) / 451L, tval = h + l - 7L * m + 114L;
   *month = (int)( tval / 31L);
   *day = (int)( tval % 31L) + 1;
}
```

This code had no H file and the source did not have a function declaration (prototype). There is no need to create an H file if you only want to use the code from Clarion. Simply add the prototype to the same .C file. Copy the function definition and add a ; to the end of the line. How do you know when to add a semicolon? Every C statement ends with semicolon, *unless* it is code block wrapped in curly braces. Data declarations using curly braces, e.g. a `struct { }`, do end with a semicolon.

I also wanted to get mangling to match so I changed the Month and Day to `short`s, and added the needed #pragmas. The revised code is shown below with a few important things bolded:

```
   #pragma save
   #pragma name(prefix=>"") // no underscore
   void easter_date( const long year, short *month, short *day);
   #pragma restore
void easter_date( const long year, short *month, short *day)
{
……..
   *month = (short)( tval / 31L);
   *day = (short)( tval % 31L) + 1;
}
```

I wrote some simple code to test this function. The above C code file I named Easter.C and my tester program I named similarly Easter.CLW. I was sure I had the code right, but I got a compile error for a missing procedure

definition. At some point I realized that Easter.C would compile to Easter.Obj, and Easter.CLW would do the same thing and overright the C code. *The lesson is do not have C and CLW files with the same name.* So I renamed my file to EasterTest.CLW and the compile worked. In the future I think I'll just add _c to the end of all my C file names.

Here's the Clarion code to call the Easter function:

```
  Map
    module('easter.c')
      easter_date( long year, <*short month>, <*short day>)
    end
  end
eYear   long
eMon    short
eDay    short
    code
    eYear = year(today())
    easter_date(eYear,eMon,eDay)
    message('Easter: ' & eMon &'/'& eDay &'/'& eYear)
```

Clarion C for Calling the Windows API

MSDN and other websites will frequently contain useful C code examples that call the Windows API. It can take some time to carefully convert such C code to Clarion. For certain situations it might be useful to leave it as C code and compile it that way. But to do that you'll need header files for Windows types, structures and functions.

The H files in the TopSpeed DevKet included a Windows.h and WinStyles.h. Remember these are from 1991 so don't try to use these files directly. They were created for compiling 16-bit Windows 3.0. They may prove helpful in converting the current Windows SDK headers provided by Microsoft. The SDK files use `Extern "C"` which I found to be a problem. I found the use of `"Extern "Pascal" {}"` in the TSDK H files to be needed to get my Windows SDK API prototypes to not have names starting with an underscore.

The Win SDK header files are large and complex with many preprocessor commands. They have many switches to allow targeting different CPUs like Alpha, Itanium, Mac or MIPS. They also deal with the API supporting ASCII and Unicode versions of functions, different Windows versions and even Visual Studio versions. Recently 64-bit and .Net have affected them. It would probably be best to start with an SDK from 1998. I unfortunately started with 2001 files before it occurred to me to dig back through my MSDN CDs.

I dove in and played with it and got some basic headers working enough to say "Hello world" in a message box and show the version of windows using the MSDN GetVersionEx sample code. I have included some tips on C programming in the file. The first four tips are to remember that C statements end with a semicolon! Also remember that C is case sensitive, and that = is an assignment, == is a compare.

There is a lot of tricky stuff in the SDK H files, and there are many includes. The files I created are included in the download in the WinTSC directory. I used the standard names, like Windows.h, but I think if I did it over I would rename them prefixed with TSCW_ to avoid any future confusion.

I was a bit ambitious in trying to get most of the Intel Windows 32-bit ASCII types defined while stripping out the rest. My goal was to have everything possible defined, so it would be possible to drop in function, equate and

structure declarations with little or no modification. As I need functions I'll add them and their required structures and equates.

I started with the SDK Windows.H and hacked back from there. It may have been a better choice to start with an empty file and add from there. Since 95% of my Windows H file is untested you probably will run into trouble, so don't assume anything works. If you are using this for production I would suggest you start with a new Windows.H and add each element as you need it, then test. I will warn you that the IDE is not very helpful in editing errors in the .H files.

Compiling and Calling C Summary

To recap, these are the steps you take to get from C source code files to compiling the C with the Clarion C compiler and calling functions from Clarion code:

1. Obtain the C source .H header file and a .C source code file and save them in your project directory or a common directory in your RED path. Do not name them the same as a CLW file. Ensure you do not have Unix end-of-line `<10>` characters as these will cause compile errors.
2. Add the .C file to a Project and see that it compiles. Obtain any required library include files.
3. Edit the .H file and wrap the prototypes in extern "C" {}
4. Create a Clarion Inc file using the .H file as a starting point
    a. Convert `struct` to `GROUP`
    b. Convert `#define` to `EQUATE`
    c. Wrap prototypes in a `MAP` and `MODULE`
    d. Convert C types to Clarion types
    e. Move the return type from the front to the end
    f. Add `RAW` attribute to each prototype
    g. Add `NAME('_name')` to each prototype
    h. Add a `CLASS, LINK('File.C')` to compile C module
5. Write a test program to verify the C code works.

I typically combine steps e, f and g by adding `,RAW,NAME('_')` to the end of all the prototypes, then cut and paste the return type, and copy and paste the procedure name after the underscore. You can also deal with external naming using other methods, or mangling as described earlier.

I hope I have inspired you so that the next time you find some C source code you need to implement in your Clarion project, you don't try to convert it to Clarion code, but instead take a shot at compiling it as C code. If you find any interesting web pages with useful C code please post them as a comment at the end of this article.

Download the source

Further Reading:

Clarion Magazine - Gordon Smith on C++ and Clarion

Clarion Documentation: The Programmer's Guide, renamed in version 6 to the Advanced Topics and Reference Guide. The chapter on Multi-Language programming is a must read, although it does have some wrong information

regarding mangling and the Pascal naming convention. The TopSpeed Project System and API Calls chapters are also useful.

Christophe Devine's website with the crypto page C source used in the article

Using C in Clarion by L3 shows how to write a function in C and include it in your Clarion project

DJGPP 32-bit C/C++ development system - This is a free IDE and compiler. The reason I mention it is the documentation page on the site has a lot useful info. There is an online C library reference.

Digital Mars C++ Compiler - The Zortech and Symantec C++ compiler are now free. Some useful information at the site. For just $42 you can have a CD with the works.

A Pascal Programmer's Guide to C - An article to teach you what you need to know to read C and understand it. Clarion is enough like Pascal to make this worth the read. There is also a part 2.

The Wikipedia contains information on encryption and hashing algorithms as well as many other topics

Ron Rivest's home page has links to articles on encryption and hashing algorithms.

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

Add a comment

- » What about calling a C# function?
- » C# should be very easy to compile and call from...

Clarion Magazine

Progress Bars for Queues: Creating An ABC/Clarion Template

by Bjarne Havnen

Published 2005-03-30

Last time I explained how to implement the `QueueProcesssManagerClass` (QPMC) in a regular window. It really wasn't that much code, but it was the type of code I am going to use very often, so I would welcome a template. In this second part of the article I will explain how to write a wrapper template for the class. The way it is done can be applied to any ABC compliant class. I have no intention of explaining every bit of the `#GROUP`s called. Rather check into the ClarionMag article "Class Wrapper Templates The Easy Way" by Lee White, as most of that information is still valid for Clarion 6.

Why a template?

A developer I knew told me that with Clarion, I should know two rules: The first is "Always see if a template exists for the specific task", and the other is "If you are going to do the job more than twice, make a template". When I first began writing templates, I used the Clarion (legacy) template chain. At first sight, these templates are a lot easier to deal with when it comes to writing your own than are the ABC templates. Using the Clarion templates I simply write some code into a handful of embeds and the task is done, while in ABC there can be more than a hundred embeds available.

I knew I could get away with a very simple template to enable the QPMC. After all, I entered code in only five embeds to get it working by hand code. It took just 10-15 minutes to transfer this code to a template using the simplest possible approach, i.e. copying embed by embed.

Every method has its pros and cons. The benefit of this approach is that is very fast, it takes little work from source to template, and even the 5.5 Template Writer generates almost correct template code for the task. However, there's a catch.

If I just enter a single comment in an embed, the template code is generated into the source. The same is the case if I use a template to generate code into different embeds, like all the code that is generated into `ThisWindow.Init`. If I construct my template the way I originally did, I will end up with all the methods generated in every procedure, regardless of the methods being used or not. If I need `ValidateRecord` once, I will have to extend my template, and that particular method will have to appear in all procedures that

use the QueueProcessManagerClass.

I want it the ABC way

In ABC, the template system shows you all the possible embeds for all the methods, and only if you embed code are the corresponding methods generated. In the embeditor every single virtual method has it's own three embeds, before and after the parent call, and data for the method. If no code is written in those embeds, the method is skipped on generation.

I can add methods through a derived local object, so it will be possible for me to enter all code in methods instead of routines, encapsulating all functionality in one single object. I want this functionality for the QPMC in both the ABC and Clarion templates. As well, as of Clarion 6 there is support for the Application Builder Classes in the Clarion (legacy) template chain. Since it is now possible to have ABC classes in the Clarion templates, any wrapper template should be built to support both chains.

Many of the ABC classes have become so complex that they are almost impossible to implement without templates in the first place, at least not in a reasonable time frame, so a lot of work has been done to standardize the implementation. Anyone who has dealt with multi-DLL applications has probably discovered that every header file in the libsrc directory is included in the data DLL. At one point a category parameter was added to the !ABCIncludefile in the header files for rarely used classes. For these classes, an extension template is necessary to tell the data application to export the class.

```
#At(%BeforeGenerateApplication)
#CALL(%AddCategory(ABC), 'thecategory')
  #CALL(%SetCategoryLocationFromPrompts(ABC),
    'thecategory','ABC','')
#EndAT
```

The QPMC is a regular ABC class, so I don't have to worry about the export, but it might be worth considering to make it a habit to include the lines above even if you choose to comment the block. I expect more and more files to be categorized in the future, so it won't be a surprise if I suddenly need them.

Dave Harms recently discussed the benefit of having code in #GROUPs, and there are an enormous number of them in the template system. Much of what I need to do to create a template wrapper involves calling specific #GROUPs. Refer to Dave's article for the background on how #GROUPs work.

Using the template wrapper #GROUPs

For my template to work the way I want to, I need to:

A. Tell the ABC templates what class I use
B. Set up the necessary prompts
C. Generate the embeds

First things first (line breaks added for readability):

```
#CALL(%ReadABCFiles(ABC))
#CALL(%SetClassDefaults(ABC),
   'QueueProcessManagerClass',
   'ThisProcess','QueueProcessManagerClass')
```

With these two lines, I refresh the list of classes and tell the template system I would like to use an instance of the QPMC called `ThisProcess`. It is only necessary to include these calls once, in the #ATSTART embed, but it is customary to also include them in the #PREPARE section in order for the developer's code to see the default values before generating the source. Note that all #GROUPs are called with ABC as first parameter, to redirect from my template chain to the ABC template chain.

In earlier versions of Clarion, I would also have to call the `%AddObjectList` and include the header file. The detection of classes used by templates has been slightly improved, so it takes less that 20 lines of template code to have a local object with all embeds. For those who have interest, the obsolete code is noted in the finished template.

Along with all the #PROMPTs I want for tuning my finished object, like prompt for queue, records per cycle, different buttons and such, I add some standard lines for enabling change of object name and possibility to derive the class

```
#Insert(%OOPPrompts(ABC))
#With(%ClassItem,'QueueProcessManagerClass')
  #Insert(%ClassPrompts(ABC))
#EndWith
```

`%OOPPrompts` setup a hidden box with two hidden prompts that the developer should never touch. `%ClassPrompts` generates rest of the prompts for class, so I end up with the standard dialog for ABC classes, as in Figure 1.

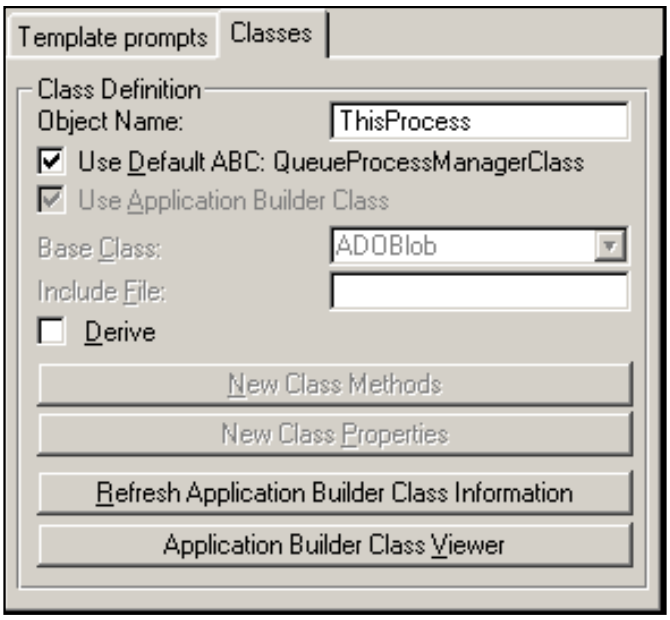

Figure 1. The ABC classes dialog

The Object Name is identified as `%ThisObjectName` in the template, and is the only prompt I need to deal with.

In order to use an object, I have to declare it.

```
#INSERT(%GenerateClass(ABC),'QueueProcessManagerClass')
```

This generates the entire `ThisProcess` object with all possible virtual methods to override. If I had used my first approach I could have more than 30 methods unconditionally declared; these are now reduced to one line.

Next is to generate the entire tree of embeds. The `#GROUP` used for this task is `%GenerateVirtuals` and is the `#GROUP` that takes most of the consideration in developing a class wrapper.

The second and third parameter for `%GenerateVirtuals` are essential to the final appearance and use of this template, the third being most important. At this point I have to decide if I want to be able to use this template more than once for any given procedure or if I can get away with a standard generation of embeds. If I go for a single instance, I can let the standard ABC group `%ProcessVirtuals` generate the embeds, and they will be known and available to other templates as `%ProcessManagerMethodCodeSection`. This will make any global template that is supposed to affect a process also affect my queue process.

If I need my template to be able to be used more than once in any given procedure, I will have to make a copy of the `%ProcessVirtuals`, change the name of the `#EMBED`s, and take `%ActiveTemplateInstance` into account. Then each object will have to have its own node in the embed tree. This is not that difficult (see Lee White's article for a detailed explanation), but it sort of diverges from the original intention of making the template.

Since I want to make a process that is similar to any other process, I decided to stick with one instance of the template per procedure, call the `%ProcessVirtuals` and try to make it look as ABC as possible.

The second parameter is more of a user issue, where you want the embeds do appear in the embed tree and how you want to identify it when you search.

The process template uses

```
Local Objects|Abc Objects|Process Manager.
```

whereas a browse uses

```
Local Objects|Abc Objects|Browse on ' & %Primary & ' using ' &
%ListControl
```

If I had gone for the multiple template, I would have to use a unique description to separate the nodes, the template instance is often used for this purpose, and if I want this to look as the process does, I stick with the process template's description.

I almost always change the name of the object from the default. After some time of development, names like

BRW1, BRW2 don't help me a bit. A name like BRWCustomers tells me a lot more, but it is not visible in the embed tree. If I had several browses on the same file, there is no obvious way to identify them expect for changing the field equate label for the list. One should treat others as you want them to treat you, so I will put the object name within the embed tree.

```
#CALL(%GenerateVirtuals(ABC),'QueueProcessManagerClass','Local Objects|Abc
 Objects|'&%ThisObjectName,'%ProcessVirtuals(ABC)')
```

Only one little piece is missing for the class to be ready for use: the parent call. This is generated by a #GROUP called %GenerateParentCall. In some earlier versions of Clarion, it was necessary to test for valid parent calls, and I have seen code to support this even in very new templates, including my own. This is no longer an issue, since the call to %ParentCallValid is done from %GenerateParentCall.

Cross template compatibility

I will now make this template compatible with both the ABC and Clarion chains.

First I will set up the necessary prompts.

```
#TAB('Template prompts')
 #Prompt('Label of queue',Expr),%Queue,REQ
 #Prompt('Records pr cycle',Expr),%RecordsPerCycle,Default(25)
 #Prompt('Timer setting',Expr),%Timer,Default(1)
 #Prompt('Message on no records',Check),%NoRecords
etc,etc
#EndTAB
```

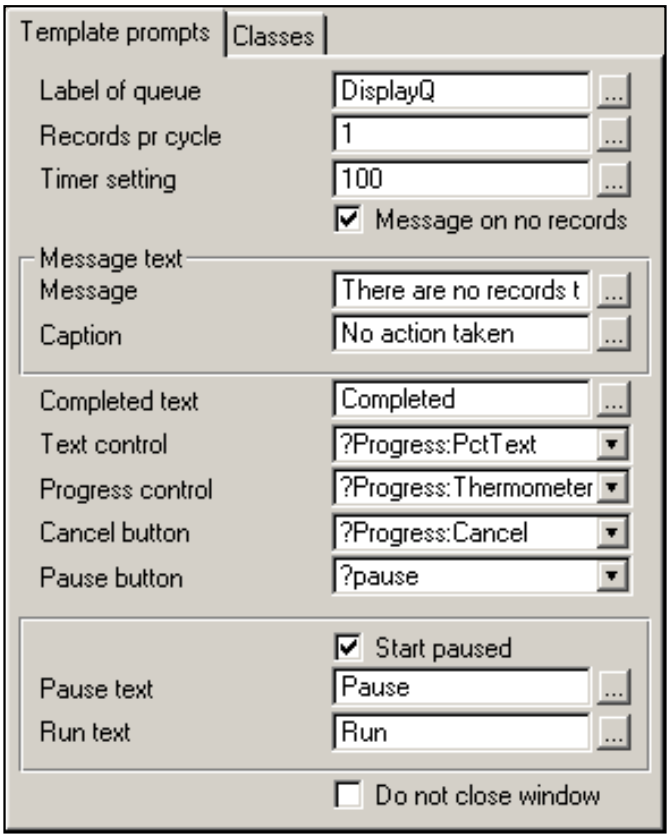

Figure 2. Setting up the prompts

I prefer to use the EXPR prompt type where I can, to make it easier to deal with the growing need for internationalization. Either plain text, an equate, or a variable is accepted. It is common to implement the check for "!" in beginning of expression and accept anything else as constant value. If I am dealing a lot with variables that are supposed to be text, I usually copy the `%StripPling` #GROUP and call it this way:

```
%ThisObjectName.SetControls(%(%StripPling(%Completed)))
```

`%StripPling` will fix any exclamation, evaluation, etc. Besides that I always try to make a template as complete as it can be. The best thing is a template that you can just add and forget.

Set the default values to the value most likely to be used. If this is done, you can extend a template with myriads of prompts, but rely on the default action to be acceptable. The PDF output template is one such example.

The rest of the template is a copy of the code from my previous article, expect for replacing `ThisProcess` with `%ThisObjectName`, and replacing other controls with the #PROMPTs. I have used regular "legacy embeds" so the template will work identically in ABC and Clarion templates..

On to the generated source code. First I intialize the object. In the ACCEPT loop I call the `TakeEvent` method, I check for no records, and I have a added possibility to start and stop the process.

```
#LOCALDATA
Progress:Thermometer BYTE
#ENDLOCALDATA
```

```
#At(%AfterOpeningWindow)
%ThisObjectName.Init(%queue)
%ThisObjectName.SetControls(%(%StripPling(%Completed))),|
  %ProgressThermometer,%ProgressCancel,%ProgressPctText)
   %ThisObjectName.SetProgressLimits(%Timer,%RecordsPerCycle)
   %ThisObjectName.Open()
   %ThisObjectName.Run()
   #If(%StartPaused And %PauseButton)
   %ThisObjectName.SetPause()
   %PauseButton{Prop:Text}=%(%StripPling(%RunText))
   #EndIf
   #If(%NoCloseWindow)
   %ThisObjectName.SetCloseWindow(False)
   #EndIf
#EndAT
#AT(%AcceptLoopAfterEventHandling)
   ReturnValue = %ThisObjectName.TakeEvent()
#EndAT
#At(%ProcessManagerMethodCodeSection,'TakeNoRecords')
      ,Priority(7500),Where(%NoRecords)
  Message(%(%StripPling(%MsgText)),|
    %(%StripPling(%MsgCaption)))
#EndAT
#At(%ControlEventHandling,%PauseButton,'Accepted')
      ,Priority(2500),Where(%PauseButton)
%ThisObjectName.SetPause()
If %ThisObjectName.IsPaused()
  ?{Prop:Text}=%(%StripPling(%RunText))
  %ThisObjectName.TakePaused()
Else
  ?{Prop:Text}=%(%StripPling(%PauseText))
End
Cycle
#EndAT
```

As I mentioned earlier, there is a little design issue in the `TakeEvent` method regarding the handling of a pause button. The code only pauses the process, it doesn't restart it. Since a regular process has a pause button that stops and starts, and is able to start paused, I have extracted the pause functionality from the base class. This class is very pure in design, and most of the properties are either private or protected, but methods are provided to read and set the properties from outside the object.

The first thing I do is call the `Setpause()` method. This method toggles pause on/off. Next I check if this call made the process pause, and if necessary I change the text according to the state of the process. The `TakePause` method is called for special handling in the event of a pause.

Now I am pretty much satisfied. I have an object where I instantly know where to put every bit of code, and I have a template interface to activate it.

Using the template

Let's say I am now going to make a batch email process where the customer will pass a queue of customer IDs. The methods that are most likely to be used are as follows:

`TakeRecord`: A valid queue element is found

`ValidateRecord`: Check for particular conditions to exclude records. For my task, I will exclude queue elements referring to a tagged customer with no email.

`TakeCompleted`: Show a message indicating how many records have actually been processed, and clean up the tag queue for those being processed.

`TakeCancelled`: Clean out the elements that have been processed.

I could design the template to handle almost any event I like, but I think the current design is sufficient. It is probably better to make another extension template to generate the necessary code for peculiar actions. The best thing about wrapper templates like this one is that they all are basically identical in design. I have several classes that I might need to fine tune for a specific task. Writing the wrapper template gives me all the flexibility I need, and even if I don't need to alter the code, I will save some work implementing the classes.

Now that the QPMC is done, I will have to hunt for another undocumented feature.

Download the source

Reader Comments

Add a comment

Clarion Magazine

Clarion Magazine's E-Books

E-books are another great way to get the information you want from Clarion Magazine. Your time is valuable; with our e-books, you spend less time hunting down the information you need. We're constantly collecting the best Clarion Magazine articles by top developers into themed PDFs, so you'll always have a ready reference for your favorite Clarion development topics.

E-book format

All our e-books are unencrypted PDFs, and are formatted to the same standards as our print books. All you need to read them is Adobe Reader or any other PDF viewer.

Free e-books

If you take out a Clarion Magazine subscription or renewal by April 8, 2005, you'll automatically receive an electronic coupon for **a free e-book**! You can verify your coupon by logging in to your My ClarionMag page.

Free updates

We're constantly updating our e-books with the latest articles. All e-book purchases come with a three month free upgrade policy; if you're also a subscriber, you get unlimited free updates, as long as you have an active subscription.

Subscriber discounts

Clarion Magazine subscribers can purchase e-books at discounted prices.

Pricing

Most e-books will be in the range of $9.95 - $19.95.

Current and upcoming e-books

E-Book	Your Price	Regular Price	All prices in US Dollars
E-Book (PDF): Mastering Clarion DLLs (version 1.1) Almost any Clarion application can benefit from being split into an EXE and one or more Dynamic Link Libraries (DLLs). This collection of articles shows all the tricks for getting the most out of DLLs, from how to easily split your applications up for easier maintenance, to calling unlinked DLLs at runtime, to rebasing your DLLs (and third party DLLs) for greatly improved load times. An essential handbook for anyone who develops large applications. View the table of contents	**$9.95**	$19.95	Add to Cart
E-Book (PDF): Learning The Clarion Language Many Clarion developers begin writing applications with the AppGen, and then find themselves wanting to do more with the Clarion environment. But learning how to write Clarion code by examining the generated code can be overwhelming. This ebook begins with an overview of the Clarion environment, and by using simple examples shows how easy it is to write Clarion code. Topics include standard Clarion data types and equates, creating procedures, list box formatting, and basic file handling techniques. View the table of contents	**$9.95**	$19.95	Add to Cart

E-Book (PDF): Learning The Clarion Template Language The real power of Clarion is its template-based code generation. Just like the shipping templates, your own custom templates increase your productivity and reduce the effort required to maintain code. And writing templates is easier than you think, as this intro-level ebook shows. Topics include template language basics, code templates, extension templates, reusable template #GROUPs, and the Template Wizatron/Writer. View the table of contents	**$9.95**	$19.95	Add to Cart
E-Book (PDF): Threading In Clarion Clarion 6 has opened up a new, exciting, and potentically confusing world of threading to Clarion developers. This collection of articles will guide you through the various new functions, classes, and techniques available to unleash the power of true threads in your applications. As a bonus, you also get Jim Kane's classic articles on using API threads with Clarion 5.x. View the table of contents	**$9.95**	$19.95	Add to Cart
E-Book (PDF): More E-Books Coming in April/May We have a number of additional e-books on the way! Upcoming topics include debugging, COM, Edit-In-Place, using C/C++ code, and ABC design notes - stay tuned!	**$9.95**	$19.95	Available April, May 2005