Clarion Magazine

For marketing your Applications and Developer Accessories or to purchase other 3rd Party Tools . . . Developer PLUS tm

Server Downtime Notice

The Clarion Magazine server will be unavailable from **12:00 p.m. EST March 31 to 6:00 a.m. EST April 1**. The server is being moved to a new facility. The web site, email, and newsgroups will be affected by this move.

PDF For February, 2003

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

Posted Monday, March 03, 2003

Next Issue March 9-15, 2003

This is an off week for ClarionMag. The next issue will appear the week of March 9-15, 2003.

Posted Wednesday, March 05, 2003

Clarion 6 EA3 Released

Clarion 6 Early Access release 3 is now available to program participants. This release includes numerous bug fixes, changes, and enchancements.

Posted Friday, March 07, 2003

Menu Buttons In Clarion

If you've ever used Intuit's Quicken software, you will have seen Menu Buttons in action. Instead of a standard menu bar across the top of the window, Quicken uses what appear to be flat, transparent buttons, which display a popup menu when clicked. Brice Schagane shows how to get the same effect in a Clarion application.

Posted Tuesday, March 11, 2003

Post-mortem Debugging: How I stopped fearing the GPF

Sometimes, when you're testing that almost completed application, it crashes. No rhyme or reason, and inspection of the last code changes reveal nothing out of place. Re-run and it crashes again. Where do you go? To the debugger for a quick post mortem. Russ Eggen shows how to track down those tricky GPFs.

Clarion Magazine **subscriptions** (online only) are $49 for six months, $95 for one year and $170 for two years.

Subscribe **Renew**

News

EasyListPrint 1.00

ShapeMaker:SMX For Buttons

qbFUSE 1.2 released

OutlookFUSE 1.2 Released

xmlFUSE - Microsoft XML And SOAP Automation For Clarion

xTransparent v1.4

R-Install Version 1.g

RDraw Diagramming Template Alpha

Fenix ASP.NET Generator Beta 1

Easy3DStyle 1.02

Enterprise System Information 3.30

Posted Friday, March 14, 2003

The Clarion Magazine GPF Challenge

This Clarion challenge is a bit different from those Clarion Magazine has run in the past. Normally, we'd be asking you to write some code that does something truly wonderful, probably using the minimum source code, and with an absolute excess of elegance and speed. But this one is about writing code that GPFs.

Posted Friday, March 14, 2003

Weekly PDF For March 9-15, 2003

All ClarionMag articles for March 9-15, 2003 in PDF format.

Posted Monday, March 17, 2003

A Naïve Look At Pre-Emption

Steve Parker knows that with Clarion 6 on the horizon, global data abuse has to be dealt with, beginning with his own applications.

Posted Thursday, March 20, 2003

PostgreSQL 101: Security Basics

Just when you think you're ready to start using PostgreSQL, you realize you can't connect to the server. Here's what you need to know about server connection security.

Posted Thursday, March 20, 2003

My First Function Library

If you've never created a function library, Alan Telford wants to talk to you. You will make your first function library. And yes, it will be important to you! You will enjoy it! You will remember it for life!

Posted Friday, March 21, 2003

The Clarion Magazine GPF Challenge Ends This Week

Time is running out on the Clarion Magazine GPF challenge - get your entry in now!

Posted Monday, March 24, 2003

Weekly PDF For March 16-22, 2003

All ClarionMag articles for March 16-22, 2003 in PDF format.

Posted Monday, March 24, 2003

Data Structures And Algorithms Part XVII - Decompression

Last time Alison Neal discussed Huffman's compression algorithm, which provides about 40% compression for text files. In this article Alison shows how to decompress the now compressed file.

Posted Thursday, March 27, 2003

Free Utility

xTipOfDay v1.6

Product Scope 32 PRO Limited Time 50% Sale

UltraTree Clarion 6 Support

EasyResizeAndSplit 1.06

Clarion Photo Gallery

CWPlus 1.02 Free Trial

Fenix ASP.NET Generator

SealSoft xDataBackupManager Pro v1.5

New Product - HTML Designer Templates

IngasoftPlus Forums

Image Set For Webmasters And Software Developers

Clarion Jobs Page

CWPlus 1.00

EasyResizeAndSplit 1.05 Demo

Easy3DStyle 1.00 Demo

CWPlus Trial Version

DEF 5507 Released

SealSoft xReplacer (TX* Manager) Build 005

Fenix Clarion ASP.NET Code Generation Demo

RADventure Intellisense

Icetips Xplore

New Image Sets

cpTracker R4

Zip App 1.1c

CWPlus Information

HTML Designer Version 1.05 B116

xDataBackupManager Pro v1.4

EasyResizeAndSplit 1.04

xReplacer (TX* Manager) For Clarion

gREGPlus v4.51 Released

Demystifying C6 Threading (Part 3)

Mutexes, semaphores, critical sections, reader/writer locks; all of these things are part of Clarion 6, and they all have to do with the new support for unlocking Clarion threads so they run like real operating system threads. Should you care? If you write any embedded code, yes, you should. Should you worry? That all depends on what kind of embedded code you write. Part 3 of 3.

Posted Friday, March 28, 2003

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

Clarioners Database

xReplacer (TXA Manager) 1.0 Beta Build #003

Newsletter Service Update

ClarionPost.com/ClarionKB.com

Search the news archive

Clarion Magazine

Next Generation Clarion Technologies
3rd Party Tools for Clarion Developers
ThinkData, Inc.

Menu Buttons In Clarion

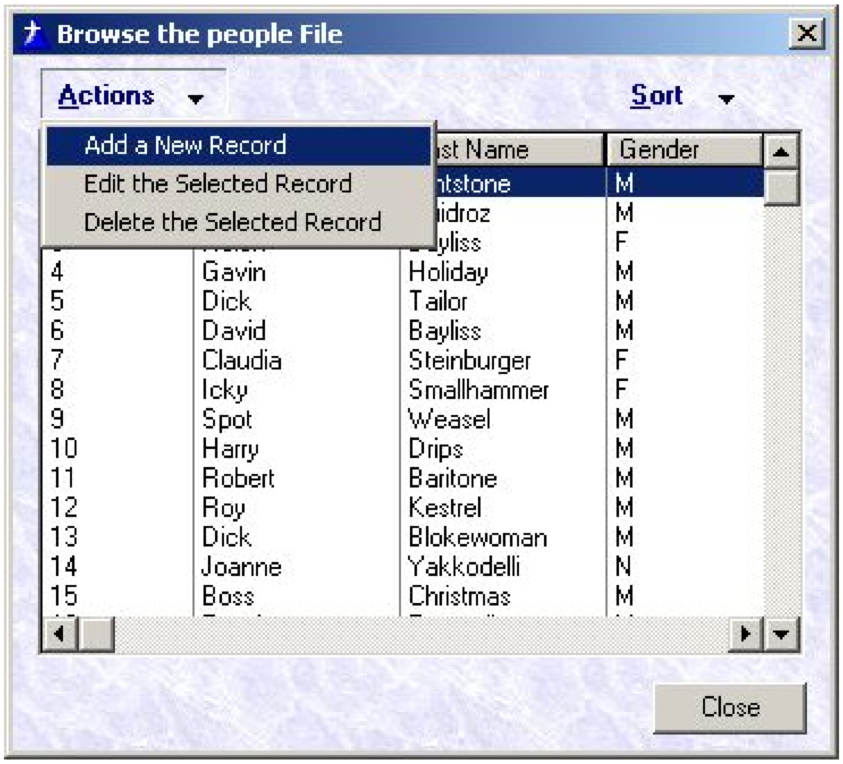
by Brice Schagane

Published 2003-03-11

Functional design may be considered the most important aspect in the creation of an application. However, in today's graphically intensive world, the importance of visual design should not be overlooked. Often times, a users first impression is formed when the application initially displays on the screen. If the application lacks a professional look and feel, the user could be turned off instantly. Once that happens, the functional design seems trivial. Menu Buttons can be used to enhance the visual design, as well as the functional design.

If you've ever used Intuit's Quicken software, you will have seen Menu Buttons in action. Instead of a standard menu bar across the top of the window, Quicken uses what appear to be flat, transparent buttons, which display a popup menu when clicked. I believe these buttons help enhance the professional look, as well as improving functionality.

I've seen many applications with multiple browses on a window, multiple sets of update controls, multiple sets of tabs, etc. In my opinion, this looks gaudy. For example, if you have multiple Insert controls, labeled simply **Insert**, a user can sometimes be easily confused as to exactly what each Insert button is used for. A more descriptive label would be helpful, but often times this is not practical, due to space limitations. By using Menu Buttons, you can not only save real estate space, but also provide a more descriptive label for each procedure. Figure 1 shows an example of an **Actions** button that leads to some standard update actions.



		st Name	Gender
		ntstone	M
		idroz	M
		yliss	F
4	Gavin	Holiday	M
5	Dick	Tailor	M
6	David	Bayliss	M
7	Claudia	Steinburger	F
8	Icky	Smallhammer	F
9	Spot	Weasel	M
10	Harry	Drips	M
11	Robert	Baritone	M
12	Roy	Kestrel	M
13	Dick	Blokewoman	M
14	Joanne	Yakkodelli	N
15	Boss	Christmas	M

Figure 1. Menu Button

Preparation

This article is accompanied by two files: Ldown.ico and WinApi.clw. You will be using both files in this tutorial. The Ldown.ico file is simply an icon. The WinApi.clw file contains Clarion source code, which I will discuss in more detail later.

Copy the Ldown.ico file into Clarion’s Images directory, and copy the WinApi.clw file into Clarion’s Libsrc directory. If a file of the same name already exists on your system, you can rename the new file to whatever you’d like. If you rename one of the included files, you should account for that change while reading this article.

You can use the example application in the dowloadable source (at the end of this article), or you can pick any Clarion application you like. Once you’ve decided on an application to use, pick a browse, any browse. Let me rephrase that: pick a browse that has **Insert**, **Change**, and **Delete** buttons.

Once you have decided on a browse procedure, open the properties dialog for the procedure. Click the **Data** button and add a new variable to the **Local Data**. Specify the following properties for the variable:

Column Name	`WinRECT`

Data Type	LIKE
Base Type	RECT

You'll actually create the RECT data type a little later. To satisfy your curiosity, here's what it looks like:

```
RECT       GROUP,TYPE
left         SIGNED
top          SIGNED
right        SIGNED
bottom       SIGNED
           END
```

Save the changes for the new variable and close the **Local Data Dialog**.

From the procedure properties dialog, open the Window Formatter by clicking the **Window** button. For each update button on the browse (**Insert**, **Change**, **Delete**), open the properties editor and select the **Hide** and **Skip** modes for each button. Save your changes.

Next, add a new button to your window, edit its properties, and make the following settings:

Text	&Actions
Use	?Btn:Actions
Justification	Right Justified
Transparent	{checked}
Skip	{checked}
Icon	Ldown.ico
Flat	{checked}

To keep your example consistent with the instructions in this article, set the button properties exactly as shown above. Set the text label of the new button to '`Actions`'. The ampersand (&) that prefixes the '`Actions`' label indicates the letter 'A' will be used as an accelerator key for the button. Define the button's **Use variable** as `?Btn:Actions`. To give the button an appearance similar to the one shown in Figure 1, set the button's display characteristics to **Flat**, **Transparent**, and **Right Justified**. Specify the included down arrow icon for the button. Select the **Skip** attribute for the button so it will not receive input focus and may only be accessed with the mouse or the accelerator key (Alt-A).

Writing the code

Next, define the functionality for the new button. To get the desired visual effect, you'll temporarily create a `GROUP` control over top of the button (this provides the depressed button appearance while the mouse pointer is not directly over the button), then execute a popup menu, and finally destroy the `GROUP` control. Under the **Actions** tab for the button, click the **Embeds** button. Select the `Accepted` event for the button, click the **Insert** button, select **Source** as the embed type, and enter the following source code:

```
!// Popup Menu: Actions
CID# = Create(0,Create:Group)
CID#{PROP:TEXT} = ''
CID#{PROP:Boxed} = TRUE
CID#{PROP:Bevel,1} = -1
CID#{PROP:SCROLL} = TRUE
GETPOSITION(?Btn:Actions,x#,y#,w#,h#)
SETPOSITION(CID#,x#,y#,w#,h#)
UNHIDE(CID#)
GetWindowRect(?Btn:Actions{PROP:HANDLE},WinRECT)
EXECUTE POPUP('&Create a New Invoice'         &|
     '|&View/Edit Selected Invoice'  &|
     '|&Delete Selected Invoice', WinRECT.Left, WinRECT.bottom)
  POST(EVENT:Accepted,?Insert)
  POST(EVENT:Accepted,?Change)
  POST(EVENT:Accepted,?Delete)
END!_EXECUTE
DESTROY(CID#)
```

When you have finished, exit the source code editor and save your changes. Verify that your embedded source code is listed just before the **Generated Code** item under the **Accepted** item. If not, adjust its position by using the **Up/Down Priority** buttons.

Notice that the source code for the button's action uses a variable labeled `CID#`. The pound sign (#) suffixing the label indicates that the variable is an implicit `LONG` variable. Implicit variables do not need to be defined in a data declaration area. Be *very* careful when using implicits, however – if you accidentally misspell a variable name, you've just created a new variable and your code probably won't work as expected. (I had to bribe the editor to let me use even these short names!)

The first five lines of code are used to create a `GROUP` control and stores its properties in the `CID#` variable. The **Boxed** and **Bevel** properties, as shown, create a depressed button effect. Next, the `GETPOSITION` function retrieves the position and dimensions (relative to the current window) of the **Actions** button. Then, the `SETPOSITION` function specifies the position and dimensions of the new `GROUP` control, making them equivalent to the **Actions** button. Once the new GROUP control has been defined, the UNHIDE procedure makes it visible, giving the **Actions** button a depressed button appearance.

The API procedure named GetWindowRect retrieves the actual screen position and dimensions of the **Actions** button. This API procedure requires two parameters. The first parameter is the handle to the button control. The second is a RECT structure that is used by the procedure to store the coordinates of the control's bounding rectangle. This procedure will also return a boolean TRUE/FALSE to indicate success/failure of the function. Since I have, uh, complete confidence in Microsoft I was not concerned with trapping the return status. However, you could easily trap for any errors that may be returned from the API procedure and take appropriate action.

Once you have the coordinates of the **Actions** button, you can use them to precisely position the popup menu. The parameters for the call to the POPUP procedure include the left and bottom coordinates of the **Actions** button. This places the top edge of the menu at the bottom edge of the button, and aligns the menu with the button's left edge, making the menu appear as though it is docked to the button. Alternatively, you could use the top and right coordinates to place the menu against the right edge of the button, while aligning it with the top.

The first parameter being passed to the POPUP procedure is a string constant containing three menu selections, one for each update button. The POPUP procedure will return an integer indicating the user's choice, or zero if the user clicks outside the menu or presses ESC (indicating no choice). For a more in-depth description of the POPUP procedure, please read the Clarion documentation.

The POPUP procedure is called from within an EXECUTE structure. The EXECUTE structure will select a single executable statement based on the value returned by the POPUP procedure. For example, selecting **Delete Selected Invoice** from the popup menu returns a value of 3, and this causes the EXECUTE structure to post an EVENT:Accepted for the **Delete** button.

Finally, after the popup menu disappears, the DESTROY procedure removes the GROUP control from the window and releases any system resources used by the control.

Defining the application

Before you can use the Menu Button, you have to define the prototypes and equates for the API call. I used SoftVelocity's Windows API INCLUDE File Constructor utility to create the WinApi.clw file included with this article. Here's the source for that file:

```
  SECTION('Equates')
HANDLE     EQUATE(UNSIGNED)
HWND       EQUATE(HANDLE)
RECT       GROUP,TYPE
```

```
left          SIGNED
top           SIGNED
right         SIGNED
bottom        SIGNED
            END
  SECTION('Prototypes')
  MODULE('Windows.DLL')
    GetWindowRect(HWND, *RECT),PASCAL,RAW
  END
```

Under **Global Properties, Embeds**, perform the following steps:

1. In the section **After Global INCLUDEs** embed the following source code:

   INCLUDE(‘WinApi.clw','Equates')

2. In the section **Inside the Global Map** embed the following source code:

   INCLUDE(‘WinApi.clw','Prototypes')

The elements necessary to call the `GetWindowRect` API function are now in place. Compile and run the application.

Conclusion

As you can see, Menu Buttons can be more visually appealing and user friendly than multiple button controls on a window. They use much less space and can provide your users with a descriptive text for each action. If your application has multiple browses on a window, you could place all update procedures under a single menu button. Then, group the menu choices for each browse by defining a separator(s) inside the menu.

In this article I’ve described how to incorporate an "Actions" menu button into an application, but there are many more possibilities. If you have an application which contains numerous Tab controls, you could hide the tabs and make use of a Sort menu button instead. You could use a Print menu button to call various reports. You could also use a View menu button to filter a browse based on the user’s selection. The list goes on and on.

Download the source

Download the example application

Brice Schagane works for the Kentucky Transportation Cabinet. He also runs a small computer company by the name of Ghost Solutions, Inc. Brice has been using Clarion since 1997.

Reader Comments

Add a comment

Very good article which I have put to use already. Is...
Bob: Yes! A set of vertical bars containing only a...

Clarion Magazine

CWPlus. Super-Tool! Easy Data Handling in Clarion IDE.
Download Now! You won't regret it!
IngasoftPlus

Ads · Comments · Writers! · Privacy · Contacts · PDFs · Freebies · Open Source

Topics > Tips/Techniques > Debugging

Post-mortem Debugging: How I stopped fearing the GPF

by Russell Eggen

Published 2003-03-14

Sometimes, when you're testing that almost completed application, it crashes. No rhyme or reason, and inspection of the last code changes reveal nothing out of place. Re-run and it crashes again.

Perhaps you add a few MESSAGE() statements – you know that is dangerous, but just this once! But MESSAGE() never fires, or the app crashes before you see the message.

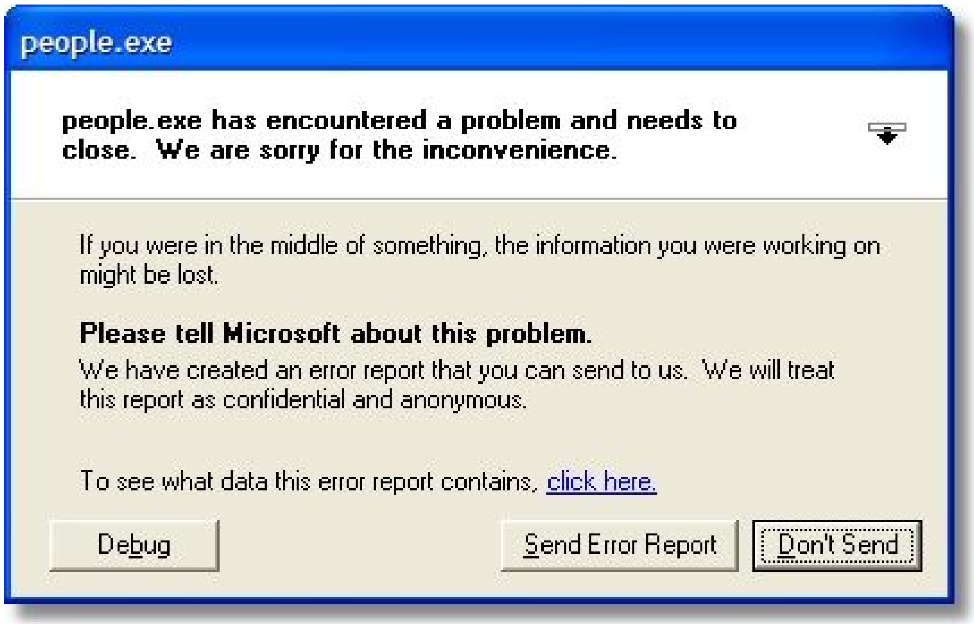
You post on the newsgroups; perhaps a colleague could reply with a clue. Not a bad tactic as the newsgroups are a good source of help. However, you feel none of the replies is applicable, or the solutions offered provide no relief.

Finally, you wind up here, Clarion Magazine! (What took you so long?)

Can the debugger really debug a crashed application? The answer is "Yes!" But if the application crashed, how can the debugger work? I'll answer that shortly, but first allow me to go through the basic steps necessary to run the debugger.

Anyone who has faced the above scenario knows that many things could cause a crash. If you are using ABC, then is it your code or the underlying classes? How are you to solve this if this means stepping through thousands of lines of ABC code? Moreover, what if you use your own classes or code libraries?

For purposes of this article, I am using a broken version of one of the shipping examples, People.APP. It runs until you try to run `BrowsePeople`. When run on Windows XP, I see the dialog in Figure 1:


people.exe

people.exe has encountered a problem and needs to close. We are sorry for the inconvenience.

If you were in the middle of something, the information you were working on might be lost.

Please tell Microsoft about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.

To see what data this error report contains, click here.

Figure 1. The GPF dialog in Windows XP.

If you inspect the detailed report of the above crash, you will find it really does not yield anything useful (unless you live in the hex world).

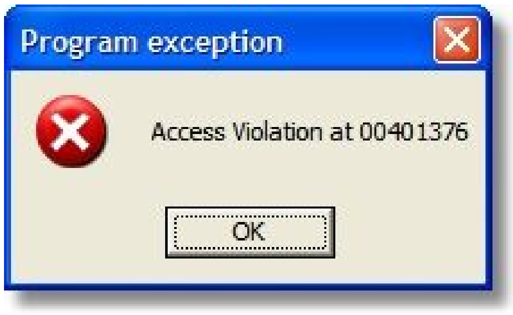
Fire up the debugger

If the Clarion debugger is the system debugger, you could press the Debug button. This launches the debugger and loads the crashed program. If the Clarion debugger is not the system debugger, (you may be using Dr. Watson or one of Microsoft's debuggers), the debug button instead shows a list of installed debuggers.

To make the Clarion debugger the system debugger, run C55DBX.EXE located in the bin folder. Go to the Setup menu and choose Install as system debugger. You may use this option with or without a program loaded.

You may also launch the application via the debugger in the IDE. Remember, these methods work only if your project is not in release mode. Use Project/Edit from the Clarion IDE or press the Project button (with your application open). Then press the Properties button and select the Debug tab to ensure that Debug mode is set to Full. There is a second option called Min meaning "minimum debug info" when running the debugger. I do not recommend this option as not enough information is included in the application to be debugged.

Since the purpose of this article is debugging a crashed application, press the Debug button to launch the Clarion debugger (provided it is the system debugger). The debugger loads, and then loads the dead (no longer running) program. This message from the debugger in Figure 2 distracts a few developers:



Program exception

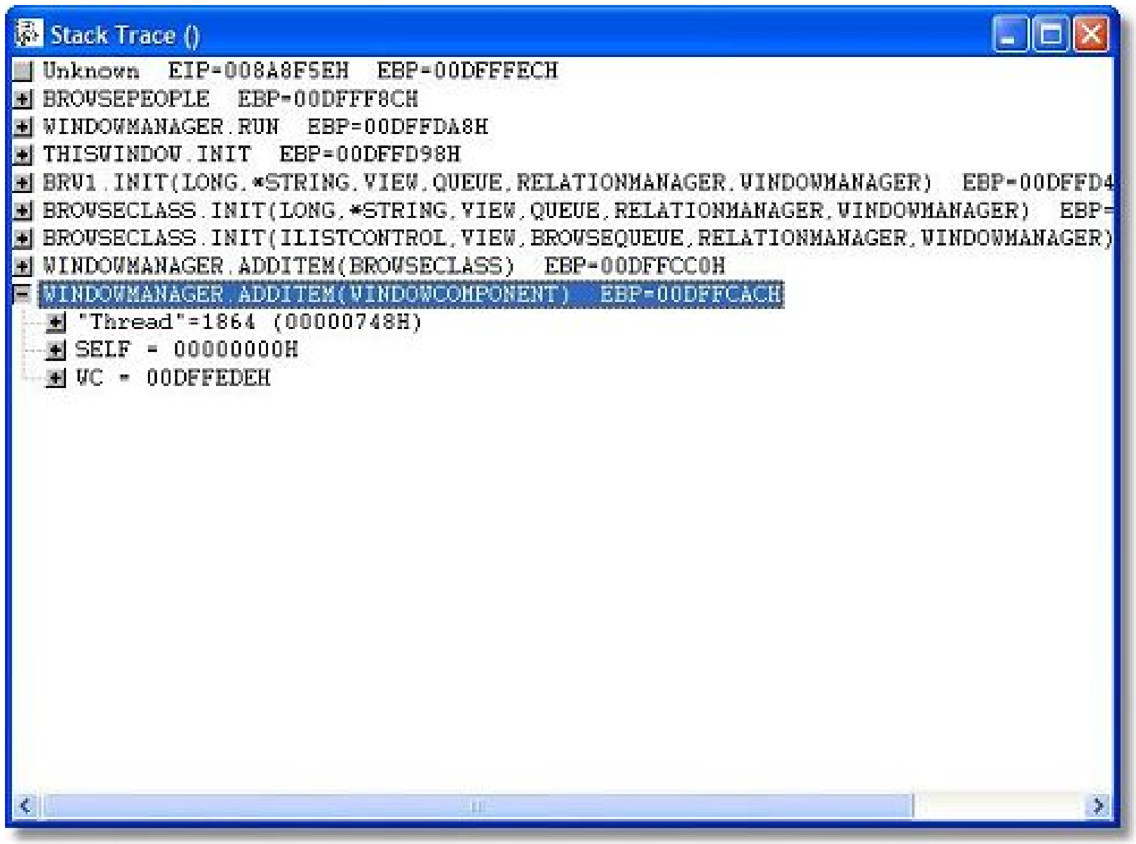
Access Violation at 00401376

OK

Figure 2. Program exception message.

One interesting thing about the above message is the address. It is the same one reported in the GPF dialog (if you looked at the details). Press OK and the debugger desktop is ready for you.

If you look at the stack trace window, you will see something like Figure 3:

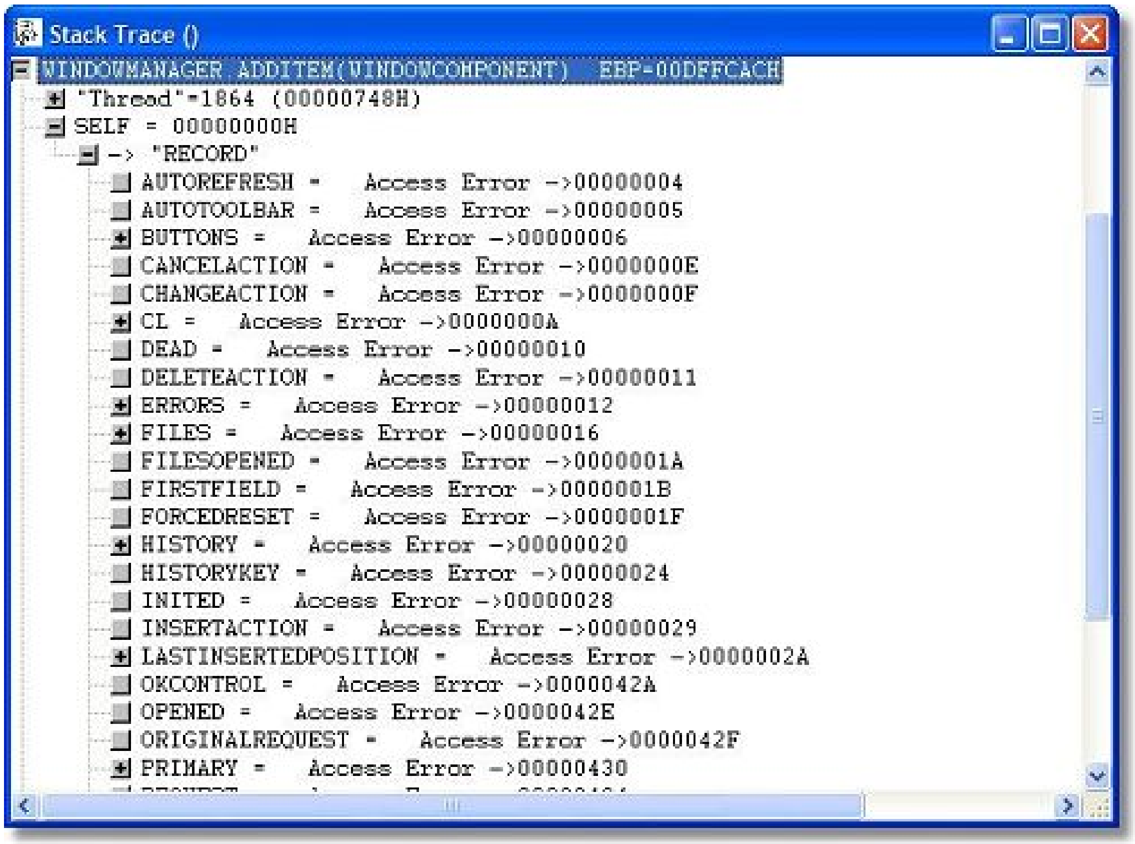


Stack Trace ()

```
Unknown   EIP=008A8F5EH   EBP=00DFFFECH
BROWSEPEOPLE   EBP=00DFFF8CH
WINDOWMANAGER.RUN   EBP=00DFFDA8H
THISWINDOW.INIT   EBP=00DFFD98H
BRW1.INIT(LONG,*STRING,VIEW,QUEUE,RELATIONMANAGER,WINDOWMANAGER)   EBP=00DFFD4
BROWSECLASS.INIT(LONG,*STRING,VIEW,QUEUE,RELATIONMANAGER,WINDOWMANAGER)   EBP=
BROWSECLASS.INIT(ILISTCONTROL,VIEW,BROWSEQUEUE,RELATIONMANAGER,WINDOWMANAGER)
WINDOWMANAGER.ADDITEM(BROWSECLASS)   EBP=00DFFCC0H
WINDOWMANAGER.ADDITEM(WINDOWCOMPONENT)   EBP=00DFFCACH
  "Thread"=1864 (00000748H)
  SELF = 00000000H
  WC = 00DFFEDEH
```

Figure 3. The stack trace window showing last method call before the crash.

Figure 3 is important as the GPF is somewhere in the Stack Trace Window. In this example, `WindowManager.AddItem(WindowComponent)` is highlighted. Notice that SELF is zero. SELF is the current object and thus it should never be zero! That is your first clue. Expand SELF by clicking on the plus button on the tree. Expand the Record node. The Stack Trace looks like Figure 4, and all the Access Error messages do confirm that there's something seriously wrong with the `WindowManager object:`



```
Stack Trace ()
WINDOWMANAGER ADDITEM(WINDOWCOMPONENT)  EBP=00DFFCACH
  "Thread"=1864 (00000748H)
  SELF = 00000000H
    -> "RECORD"
      AUTOREFRESH =   Access Error ->00000004
      AUTOTOOLBAR =   Access Error ->00000005
      BUTTONS =   Access Error ->00000006
      CANCELACTION =   Access Error ->0000000E
      CHANGEACTION =   Access Error ->0000000F
      CL =   Access Error ->0000000A
      DEAD =   Access Error ->00000010
      DELETEACTION =   Access Error ->00000011
      ERRORS =   Access Error ->00000012
      FILES =   Access Error ->00000016
      FILESOPENED =   Access Error ->0000001A
      FIRSTFIELD =   Access Error ->0000001B
      FORCEDRESET =   Access Error ->0000001F
      HISTORY =   Access Error ->00000020
      HISTORYKEY =   Access Error ->00000024
      INITED =   Access Error ->00000028
      INSERTACTION =   Access Error ->00000029
      LASTINSERTEDPOSITION =   Access Error ->0000002A
      OKCONTROL =   Access Error ->0000042A
      OPENED =   Access Error ->0000042E
      ORIGINALREQUEST =   Access Error ->0000042F
      PRIMARY =   Access Error ->00000430
```

Figure 4. The SELF record is invalid.

If you right-click on **WindowManager.AddItem(WindowComponent)**, you are presented with a pop-up menu as shown in the Figure 5:

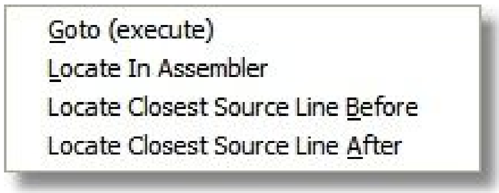
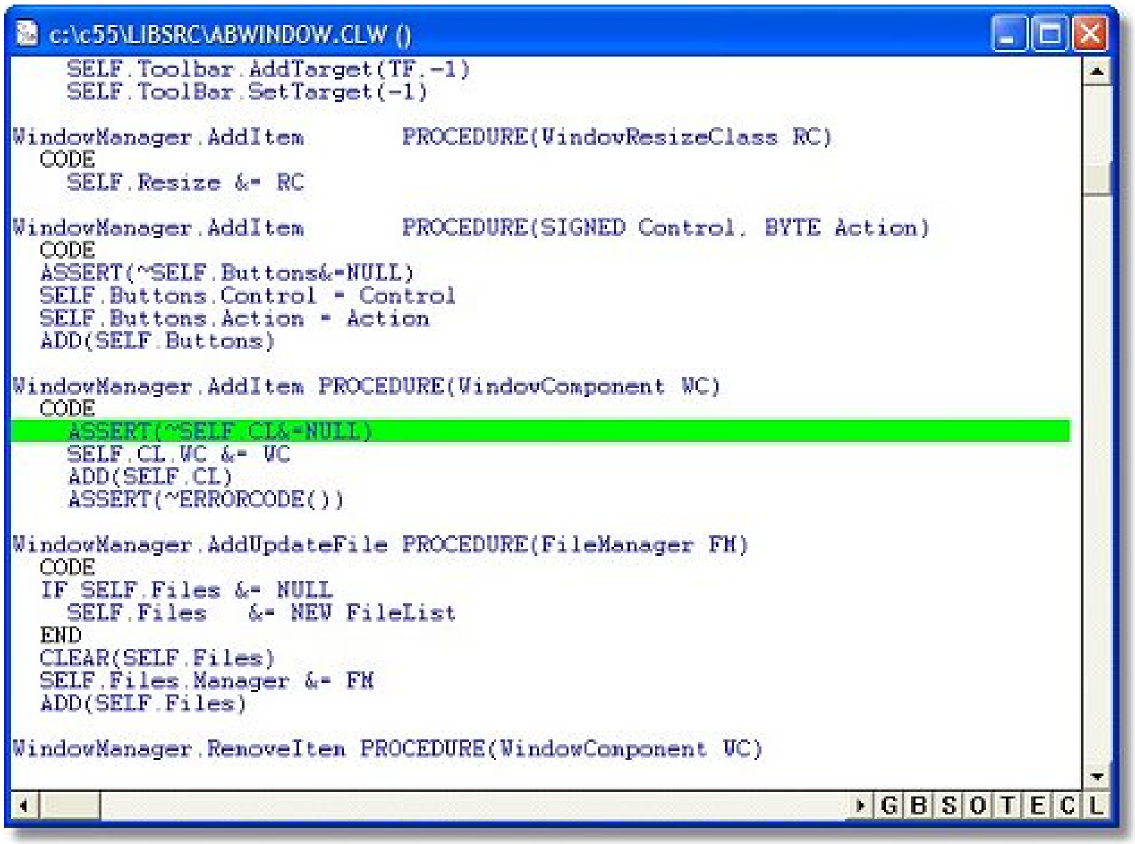

Goto (execute)
Locate In Assembler
Locate Closest Source Line Before
Locate Closest Source Line After

Figure 5. The stack trace pop-up menu.

When I debug like this, I choose Locate Closest Source Line Before. This loads the source (if not already loaded) and shows the line last executed. Figure 6 shows the source window:



```
    SELF.Toolbar.AddTarget(TF,-1)
    SELF.ToolBar.SetTarget(-1)

WindowManager.AddItem       PROCEDURE(WindowResizeClass RC)
  CODE
    SELF.Resize &= RC

WindowManager.AddItem       PROCEDURE(SIGNED Control, BYTE Action)
  CODE
  ASSERT(~SELF.Buttons&=NULL)
  SELF.Buttons.Control = Control
  SELF.Buttons.Action = Action
  ADD(SELF.Buttons)

WindowManager.AddItem PROCEDURE(WindowComponent WC)
  CODE
    ASSERT(~SELF.CL&=NULL)
    SELF.CL.WC &= WC
    ADD(SELF.CL)
    ASSERT(~ERRORCODE())

WindowManager.AddUpdateFile PROCEDURE(FileManager FM)
  CODE
  IF SELF.Files &= NULL
    SELF.Files   &= NEW FileList
  END
  CLEAR(SELF.Files)
  SELF.Files.Manager &= FM
  ADD(SELF.Files)

WindowManager.RemoveItem PROCEDURE(WindowComponent WC)
```

Figure 6. Source window with last executed source highlighted.

Remember, the pop-up menu was to show the line before the GPF, therefore the line of code for the assert did execute. But the stack trace showed that the WindowManager itself had not been allocated any memory. How can you call a method on a non-existent object? You can't, at least not without a GPF. This code triggered the crash, but it did not cause it.

The Stack Trace window is your best tool here and the only tool from the debugger that is driving your investigation. You cannot step through source at any level as that requires a running program and this one is as dead as a doornail.

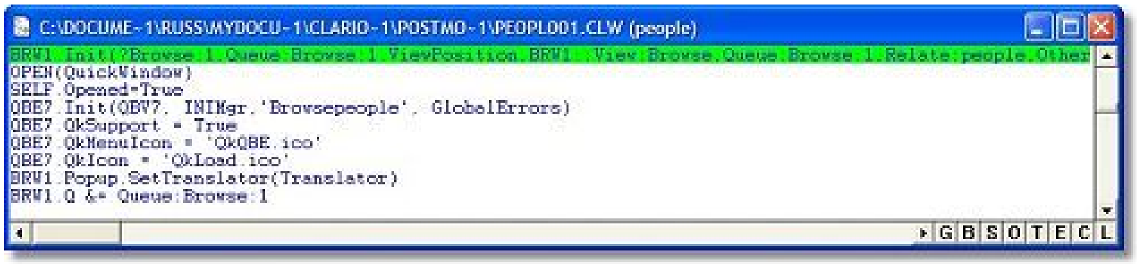
Collapse the node as it can only confirm that something has indeed gone wrong. Expand the next node up, `WindowManager.AddItem(BrowseClass)`. This has the same problem as before, SELF has errors. If you want, you could call the pop-up menu and inspect that source. However, it is the same dead-end as before. This is not the cause of the GPF as other procedures use this code with no problem.

On to the previous node, `BrowseClass.Init`. If you expand this node, something is different. Here, `SELF` is valid, `WM` is not! Expand `WM` as before and you will see the same errors. This means that this method was passed a bad WM (it is the last parameter).

To be on the safe side, inspect the previous node, another `BrowseClass.Init` method (different parameters). It too has an invalid `WM` parameter.

As does `BRW1.Init`, but the previous node up, `ThisWindow.Init` all its values appear fine. This is revealing! Clearly the cause of the error is in one of these two methods. `BRW1.Init` may get something bad from `ThisWindow.Init`. In addition, the debugger has already shown it is a bad `WM` value.

Right click on ThisWindow.Init and choose Locate in Source. This shows the call to `BRW1.Init()`. At this point, you also know that the Window Manager parameter is invalid. Inspect the call, as it exists in the source:

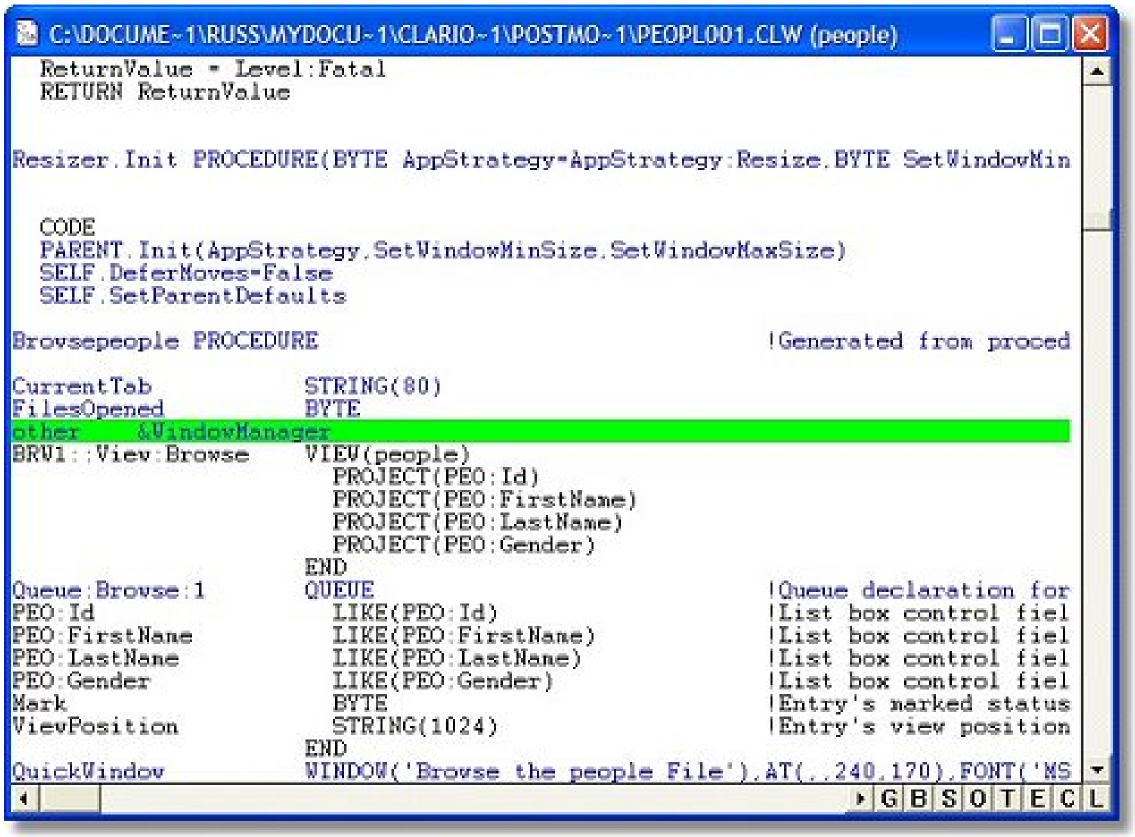

C:\DOCUME~1\RUSS\MYDOCU~1\CLARIO~1\POSTMO~1\PEOPL001.CLW (people)

```
BRW1.Init(?Browse:1,Queue:Browse:1.ViewPosition,BRW1::View:Browse,Queue:Browse:1,Relate:people,Other
OPEN(QuickWindow)
SELF.Opened=True
QBE7.Init(QBV7, INIMgr,'Browsepeople', GlobalErrors)
QBE7.QkSupport = True
QBE7.QkMenuIcon = 'QkQBE.ico'
QBE7.QkIcon = 'QkLoad.ico'
BRW1.Popup.SetTranslator(Translator)
BRW1.Q &= Queue:Browse:1
```

Figure 7. ThisWindow.Init calling BRW1.Init

Figure 7 shows all parameters to `BRW1.Init`, so you may need to horizontally scroll or resize the width of the window to see everything.

The `Other` parameter is the `WindowManager`'s spot in the parameter list. The compiler certainly did not complain about `Other`. This means it must be declared and of the proper type. While you are still in the source window press **F**. This opens a Find dialog. Enter `other`. You may get a message about it not being found and do you want to look for the value from the start of the file. If so, press OK. By now, the debugger should find the `Other` declaration.



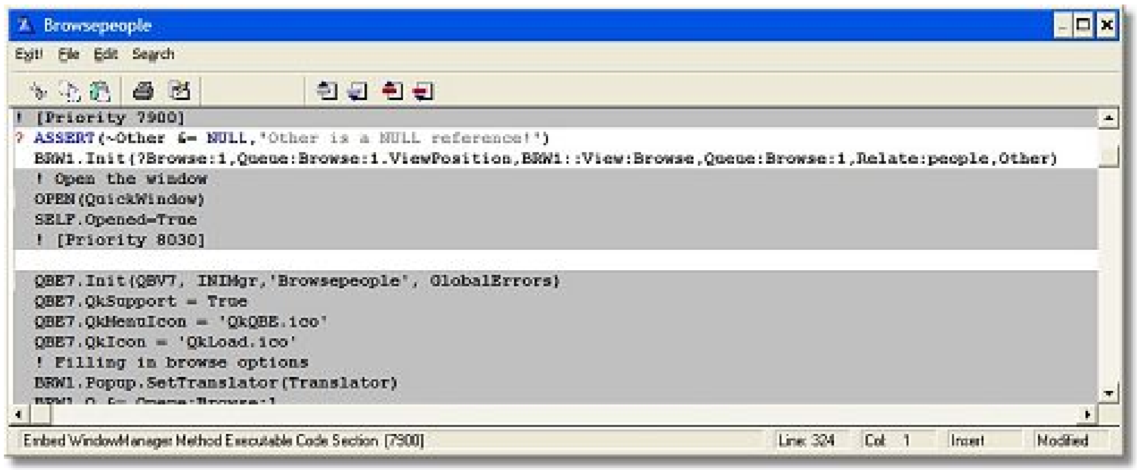
```
C:\DOCUME~1\RUSS\MYDOCU~1\CLARIO~1\POSTMO~1\PEOPL001.CLW (people)
  ReturnValue = Level:Fatal
  RETURN ReturnValue


Resizer.Init PROCEDURE(BYTE AppStrategy=AppStrategy:Resize,BYTE SetWindowMin


  CODE
  PARENT.Init(AppStrategy,SetWindowMinSize,SetWindowMaxSize)
  SELF.DeferMoves=False
  SELF.SetParentDefaults

Browsepeople PROCEDURE                                   !Generated from proced

CurrentTab           STRING(80)
FilesOpened          BYTE
other    &WindowManager
BRW1::View:Browse    VIEW(people)
                       PROJECT(PEO:Id)
                       PROJECT(PEO:FirstName)
                       PROJECT(PEO:LastName)
                       PROJECT(PEO:Gender)
                     END
Queue:Browse:1       QUEUE                               !Queue declaration for
PEO:Id                 LIKE(PEO:Id)                      !List box control fiel
PEO:FirstName          LIKE(PEO:FirstName)               !List box control fiel
PEO:LastName           LIKE(PEO:LastName)                !List box control fiel
PEO:Gender             LIKE(PEO:Gender)                  !List box control fiel
Mark                   BYTE                              !Entry's marked status
ViewPosition           STRING(1024)                      !Entry's view position
                     END
QuickWindow          WINDOW('Browse the people File'),AT(,,240,170),FONT('MS
```

Figure 8. The Other declaration.

The `Other` variable is a reference of type `WindowManager`. That is reasonable. Use the Find function again. There must be a reference assignment in this procedure and you must ensure that the assignment is correct. However, the Find function returns to `BRW1.Init()`. This can mean only one thing; there is no reference assignment, which means `Other` is `NULL`. This is the cause of the GPF, and it cannot be traced to one line of code as the code is missing from the procedure!

Therefore, a line of omitted code caused the GPF. However certain you may be about this, prove this is the case. You are done with the debugger at this point, so close the debugger.

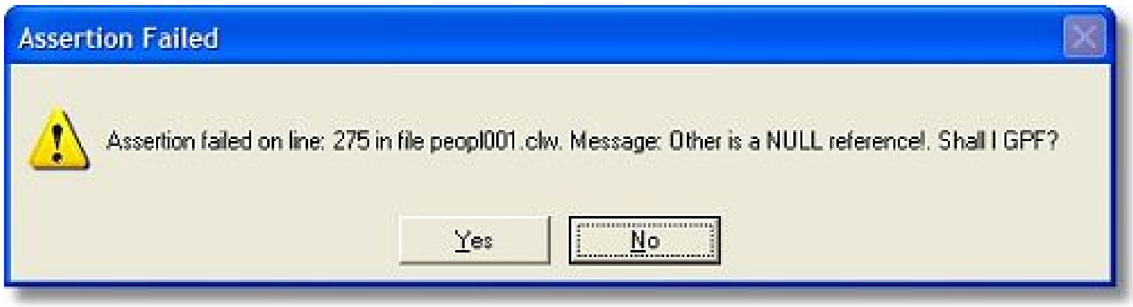
Open the `BrowsePeople` procedure with the Embeditor. You do this by highlighting the procedure and right-click. From the pop-up menu, choose **Source**. Add the `ASSERT` as shown in Figure 9. Note the ? in column 1 - this means this line of code is only compiled in when the program is in debug mode.



```
! [Priority 7900]
? ASSERT(~Other &= NULL,'Other is a NULL reference!')
  BRW1.Init(?Browse:1,Queue:Browse:1.ViewPosition,BRW1::View:Browse,Queue:Browse:1,Relate:people,Other)
  ! Open the window
  OPEN(QuickWindow)
  SELF.Opened=True
  ! [Priority 8030]

  QBE7.Init(QBV7, INIMgr,'Browsepeople', GlobalErrors)
  QBE7.QkSupport = True
  QBE7.QkMenuIcon = 'QkQBE.ico'
  QBE7.QkIcon = 'QkLoad.ico'
  ! Filling in browse options
  BRW1.Popup.SetTranslator(Translator)
```

Figure 9. Adding an assert before the BRW1.Init call.

Close and save everything. Now run the program again. You now see this message:



Assertion failed on line: 275 in file peopl001.clw. Message: Other is a NULL reference!. Shall I GPF?

Figure 10. The assert message fires, along with the optional message.

This proves the GPF is caused by a `NULL` parameter to `BRW1.Init()`, which actually happens deep in the ABC library.

So what is this `Other` declaration, really? It’s just a way to show how to use the debugger. There never was supposed to be an `Other` that got initialized and was used (it was put there by a certain meddling editor). But the same kind of thing can easily happen in your own code when you attempt to use an object which you haven’t initialized.

Summary

The debugger can breathe life into dead, crashing programs. In this case, the problem was not with existing code, but omitted code.

Another good technique is to use `? ASSERT` statements that are valid in debug mode. They work just like `MESSAGE` and `STOP` (except asserts are safe) and you can leave them in your code. Once you set your project to release mode, the asserts are not linked in. If there is an assertion in release mode, the program GPFs without the invitation to do so.

It is up to you if you wish to send your customer a debug version of his application. Your application may run fine on your machine, but not at the customer site. You could ship debug versions of select procedures as you can set individual modules to debug. You do this by highlighting a module in the project editor, then press the **Properties** button. Just change the debug mode from default to full and then ship the EXE or DLL to your customer and have them duplicate the steps and report any messages to you.

You can never have too many ASSERT statements. If you are having a problem with a bit of code and you still do not wish to use the debugger (tsk-tsk!), then use ASSERT statements. You do have to code the condition you think is happening and the optional test message. It is a great tool to test your theory as well as a wonderful support tool.

A GPF is like a letter from the IRS – you never want to get one. On those occasions where you do get a GPF, you can use it to tell you where the error is, with the debugger being your guide.

Download the source

Russ Eggen has been using Clarion since 1986. Until about 1996, he was using it for business applications, mostly accounting programs. Afterwards he joined Topspeed as a consultant, and later as an instructor. He was a founding member of SoftVelocity when that company formed from Topspeed in May 2000. He left SoftVelocity in January 2001 and now works for his own company, RadFusion Inc. He still teaches and lectures, and is currently working on a new book and setting up a local Clarion classroom. Russ enjoys flying, scuba, and applied philosophy, and with great effort you might coax him into political discussions.

Reader Comments

Add a comment

Yes, I can see how that works when the problem exists in...
Sed, Still the same, except you'll have a shorter debug...

Clarion Magazine

Reborn Free CLARION online

Topics

Clarion News

Published 2001-11-21

EasyListPrint 1.00

EasyListPrint (ELP) is a set of classes and templates for automatic creation of tabulated reports, using queues or lists (for example, automatic printing of BrowseBox contents). Creates both standard WMF reports, and editable RTF reports. You may customize Title, Header, Footer, Total sections (fonts, alignments etc), number of pages. Widths of columns will be automatically calculated during printing if they are not defined in the settings. Requires Clarion 5.0b or Clarion 5.5, ABC or Legacy class template, 32 bits only.

Posted Monday, March 31, 2003

ShapeMaker:SMX For Buttons

Logic*Central has released ShapeMaker:SMX for buttons. This product lets you transform your existing Clarion buttons to visually enhanced, custom-shaped buttons. You keep your existing embed code or add code to standard Clarion button embeds. ShapeMaker:SMX is on sale at $99 (for a short time only).

Posted Monday, March 31, 2003

qbFUSE 1.2 released

ThinkData has released Version 1.2 of qbFUSE. This version features improved support for COM SafeArrays in the Plugware COM layer.

Posted Monday, March 31, 2003

OutlookFUSE 1.2 Released

ThinkData has released Version 1.2 of OutlookFUSE. This version features improved support for COM SafeArrays in the Plugware COM layer and new features in the example applications. The demonstration applications have been updated.

Posted Monday, March 31, 2003

xmlFUSE - Microsoft XML And SOAP Automation For Clarion

ThinkData, Inc. has released xmlFUSE, a native COM (Component Object Model) automation interface to the Microsoft XML (Extensible Markup Language) 4.0 SDK and the Microsoft SOAP (Simple Object Access Protocol) 3.0 SDK using Clarion 5.5 and Clarion 6. xmlFUSE relies on the professional version of the Plugware COM classes, which usher in a new era of stability and performance when writing COM automation interfaces in Clarion. This one product provides the facility for creating and parsing XML documents using DOM (Document Object Model), SAX (Simple API for XML), XSD (XML Schema Definition Language), and SOM (Schema Object Model). You can also use XSL (Extensible Stylesheet Language) to format and transform your XML documents and the MS XML HTTP library included in the MS XML 4.0 SDK will allow you to send and retrieve documents over the Internet. xmlFUSE contains numerous examples for using the XML and XML HTTP portions of the product. In addition to all of the XML functionality, xmlFUSE provides a complete native Clarion wrapper around the MS SOAP 3.0 SDK. This synergy between XML and SOAP in the same product allows a developer to interface with XML Web Services and process their requests and responses without purchasing additional tools. XML Web Services are growing in popularity and sites such as XMethods (http://www.xmethods.com) provide listings of dozens of services you can integrate into your Clarion application. The xmlFUSE product ships with complete source code to all of the class wrappers, Plugware COM layer, and example applications. Price is $249.00 USD.

Posted Monday, March 31, 2003

xTransparent v1.4

xTransparent 1.4 contains a small template bugfix.

Posted Monday, March 31, 2003

R-Install Version 1.g

RInstall is a Clarion for Windows Template that allows a developer to implement the following measures into applications created with the ABC template chain: Provide an evaluation(demo) or registered status to an application; Provide for setting time duration of demo/evaluation period; Provide for automatic serial number generation during installation; Provide for unlock key registration of the application; Provides and option to record limit tables during the demo period; Suspend program execution after demo period has expired; Provide copy protection for application. (fixes installed application to installation computer); Provides for unlock code generation by developer/reseller/distributor; Provides a mechanism to either disable or hide controls during the evaluation period. Available at ClarionShop.

Posted Monday, March 31, 2003

RDraw Diagramming Template Alpha

The RDraw template gives Clarion applications flow diagramming capabilities similar to those available in products such as Visio and Smartdraw. An alpha will be available soon.

Posted Monday, March 31, 2003

Fenix ASP.NET Generator Beta 1

RadVenture has released Beta 1 of the Fenix ASP.NET Generator, a new template set that, instead of generating Clarion code, generates VB.NET code for ASP.NET Applications. The Beta Program is open to a limited amount of users - subscribe now by calling RADventure sales at (+31)(0) 346 29 09 07 or by sending an e-mail to sales@radventure.nl. The Price of the Beta version is $499 (€475). Beta participators will receive the Gold Release at no charge. In the first three months the price of the Gold Release will be $699 (€660), afterwards the regular price will be $899 (€850).

Posted Monday, March 31, 2003

Easy3DStyle 1.02

New in Easy3DStyle 1.02: Window background color; Auto select the next tab after last field on the tab is tabbed off; Styles setting: new line height for List controls, new line height for combo controls, selected color and color for required fields, selected color for list/drop list controls, selected color for combo/drop combo controls; Several bug fixes.

Posted Monday, March 31, 2003

Enterprise System Information 3.30 Free Utility

AIDA32 is a professional system information, diagnostics and benchmarking program for Windows platforms. It extracts details of all PC components. It can display information on the screen, print it, or save it to file in various formats like HTML, CSV, or XML. For corporate users, AIDA32 Enterprise offers command-line switches, network audit and audit statistics, remote system information and network management.

Posted Saturday, March 29, 2003

xTipOfDay v1.6

New in xTipOfDay version 1.6: Windows XP support - if you turn "Program will work under Windows XP" on the General tab then all checkboxes and options will be without Transparent attribute., and the Windows XP manifest resource will be automatically compiled into your program. The screenshots have also been updated so you can see how xTipOfDay looks under Windows XP. For Registered Users! Install Kit password was changed and will be sent to you via email.

Posted Saturday, March 29, 2003

Product Scope 32 PRO Limited Time 50% Sale

Encourager Software is having a limited time 50% sale on Product Scope 32 Pro. The program is used to create and update the Clarion Third Party Profile Exchange. You can use the Viewer Version (freely available to use, no registration fee) of Product Scope 32 PRO to display the downloadable data files (or view the online profile version), but if you'd like to edit the profile exchange and have extended features, you may want to consider purchasing the Spreadsheet or Unlimited Editing versions at substantial savings.

Posted Saturday, March 29, 2003

UltraTree Clarion 6 Support

UltraTree Platinum is now shipping with full Clarion 6 support. UltraTree Clarion 5.5 to Clarion 6 migration is a straight recompile, link and go. No manual changes required. There are no known issues.

Posted Monday, March 24, 2003

EasyResizeAndSplit 1.06

EasyResizeAndSplit 1.06 is now available. Changes in this version include: Three-status image split control (normal-selected-pressed); possibility to change strategy both in window formatter and in the properties dialog for extension template; default strategy settings for LIST and BUTTON controls; Small modifications in the template; The locking of all events, excluding EVENT:SplitMove, at dynamic Split control moving; Positions/sizes calculations during window size. This is a free download for all registered users.

Posted Monday, March 24, 2003

Clarion Photo Gallery

The Clarion Developers gallery is now in its fourth year and has over 180 entries. Anyone who is a Clarion Developer can join. The Gallery has three purposes: 1) To put a face to the people we chat to in newsgroups and on discussion boards; To help strengthen the sense of community among Clarion developers; 3) To provide a place for developers to indicate they are looking for work (Sterling Data also hosts the Clarion Skill Pool).

Posted Monday, March 24, 2003

CWPlus 1.02 Free Trial

A free trial version of CWPlus 1.02 is available. CWPlus is an auxiliary integrated tool for Clarion for Windows IDE which simplifies manual coding using analogues of the most important elements of Microsoft IntelliSense technology. Features include: Pop-up help about the identifier; Help on the prototype of edited procedure; The actual list of identifiers with an opportunity to choose the necessary identifier and to insert it into the text. Requires Clarion 5.0b or Clarion 5.5F or higher, ABC or Legacy.

Posted Monday, March 24, 2003

Fenix ASP.NET Generator

RadVenture's new Fenix ASP.NET Generator is a set of templates that, instead of generating Clarion code, generate VB.NET code for ASP.NET applications. The product is built on the concept of a three tier design: user interface, business object, and database access layers.

Posted Monday, March 24, 2003

SealSoft xDataBackupManager Pro v1.5

SealSoft has released xDataBackup Manager Pro v1.5. New in this version: Windows XP support; Updatewd xDBManager Pro screen shots; Updated documentation.

Posted Monday, March 17, 2003

New Product - HTML Designer Templates

Due to requests received, the HTML Designer template set has been split off the WYSIWYG HTML Designer product to allow developers to create their HTML help links in their applications and use any Microsoft compliant HTML Help IDE to generate the help files. With the HTML Designer templates you can convert an existing Winhelp application to HTML Help without any coding or changes in the application. The HTML Help templates makes use of the existing Winhelp context strings as is. All windows and controls have a context sensitive help. Controls can also be called as anchors on window screens or in HTM pages of their own. This template set works for all versions of Clarion for Windows, (32 bit apps only). Features include: Export and compile a "barebones" html help project, making use of TIP and MSG information for default contents generation; Import into HTML Designer, Microsoft Help Workshop, or any other MS HTML Help compliant editor to customize the help project; Handle single exe or multiple DLL applications. Available from ClarionShop.

Posted Monday, March 17, 2003

IngasoftPlus Forums

Support forums are available for all IngasoftPlus products, including EasyExcel, EasyReport, EasyMultiTag, EasyVersion , EasyDocker , EasyAnimation, The chSTD library, EasyAutoEntry, EasyResizeAndSplit, Easy3DStyle, CWPlus and EasyWinIco (freeware).

Posted Monday, March 17, 2003

Image Set For Webmasters And Software Developers

Jesus Moreno has released a new image set oriented to webmasters and great for software developers. Each group contains images for a specific subject. These are not actually icons, but high resolution images that can be resized using any standard program such a PaintShop Pro©. Each set is distributed in two formats, one that will contain all layers in the image, preserving the transparency, so that you can modify, and a flat image with a shadow that is ready to use as is.

Posted Monday, March 17, 2003

Clarion Jobs Page

This page includes a list of Clarion programming positions.

Posted Monday, March 17, 2003

CWPlus 1.00

CWPlus is an auxiliary integrated tool for Clarion for Windows IDE which simplifies manual coding using analogues of the most important elements of Microsoft IntelliSense technology. Features include: Pop-up help about the identifier; Help on the prototype of edited procedure; The actual list of identifiers with an opportunity to choose the necessary identifier and to insert it into the text. Requires Clarion 5.0b or Clarion 5.5F or higher, ABC or Legacy.

Posted Monday, March 17, 2003

EasyResizeAndSplit 1.05 Demo

A new demo is available for EasyResizeAndSplit 1.05.

Posted Monday, March 17, 2003

Easy3DStyle 1.00 Demo

A new demo is available for Ingasoftplus's Easy3Dstyle. This product lets you: Change the appearance of the main window controls; Add more functionality to Entry fields; Add the new HyperLink control using standard prompt or string controls. Compatible with the standard WindowResize template and EasyResizeAndSplit (ver 1.05 and higher). Requires Clarion 5.0b or Clarion 5.5, 32 bits only. ABC or Legacy.

Posted Monday, March 17, 2003

CWPlus Trial Version

CWPlus will available as a trial version with the limitation that the CWPlus Populate Box will be called no more than 20 times during one session of Clarion. To remove all limitation register the program.

Posted Monday, March 17, 2003

DEF 5507 Released

DEF version 5507 is now available. Changes include: DATA, Groups, Queues, and Equates can now be generated locally with the PROC or MODULE switches; ITEMIZE has been added as an option to the EQUATES switch; ADDKEYFIELD is a new feature which lets you specify an additional field be added as a key component - useful for maintaining an archive system with version numbers; ADDFIELD is a new feature which lets you add fields to the generated record structure - useful for adding a LONG, Style column to a DATA,QUEUE structure used for a browse.

Posted Monday, March 17, 2003

SealSoft xReplacer (TX* Manager) Build 005

SealSoft Update xReplacer (TX* Manager) v1.0 prerelease Build #005 is now available. New in this version: Ctrl-Shift-C to copy text of embed to Clipboard; Some changes in downgrade algorithm for more accurate translation TXA to C5 format; Command line parameters; Bug fixes. New demo and install available.

Posted Monday, March 17, 2003

Fenix Clarion ASP.NET Code Generation Demo

RadVenture has been working on .NET Generation for WebForms (ASP.NET). The product name is Fenix and a demo site is available. If anybody is interesting in receiving more information (or joining the beta program) please email Sebastian Talamoni (sebastian.talamoni@radventure.nl).

Posted Monday, March 17, 2003

RADventure Intellisense

The RADIntelliSense utility gives you the opportunity to accelerate text editing. Clarion commands, Dictionary, local and global data, Classes and windows controls are all available from the RADIntelliSense selection window. It works in the embeditor, source editor and in all external source editors of your choice Notepad, UltraEdit etc). It also performs code restructure (reformat your if-then-else-end and loop blocks, etc) and provides mousewheel support. RADIntelliSense works on Windows NT 4.0, Windows 2000 and Windows XP and it isn't limited to any version of Clarion for Windows.

Posted Monday, March 17, 2003

Icetips Xplore

Icetips Software has taken over Brian Staff's Xplore templates, which are extension templates for ABC browses but can also be used on listboxes that use regular queues. The templates and classes add a lot of end-user options to the listboxes, for example sorting the browse by clicking on the header, changing header text, justification for both data and header, hide columns etc. The Icetips Xplore templates also add the ability to create a graph from data in a column and print the data in the browse as it appears in the listbox. The price for the Icetips Xplore is US$249.00 and it is available from IceTips and from ClarionShop.

Posted Monday, March 17, 2003

New Image Sets

Gitano Software has released three new specialty image sets. These are not actually icons, but high resolution images that can be resized using any standard program such a PaintShop Pro(c).

Posted Monday, March 17, 2003

cpTracker R4

cpTracker Silver R4 is now available for download. There will be a few more enhancements but currently the main priority is to create the documentation and online help for the Gold release (Version 1.0).

Posted Monday, March 10, 2003

Zip App 1.1c

Zip App 1.1c fixed a problem with the ICO option not working correctly.

Posted Monday, March 10, 2003

CWPlus Information

Some early information on CWPlus from Ingasoftplus is now available. This product gives the IDE helpful popup windows similar to Microsoft's Intellisense technology.

Posted Monday, March 10, 2003

HTML Designer Version 1.05 B116

This update to HTML Designer is the last where any new procedures, processes or functions will be added for HTML Designer Version 1. Changes in this release: Import previously created HTML help projects; Preview Clarion screens; Final update of this previewer will require a change in the HTML Designer templates to extract all the images and icons used to the Help directory. For Projects making use of cwHH HTML help template, an extention template will be provided to do the extraction. Final release date of HTML Designer Version 1 is expected to be within the next two months, after all the bugs in the new processes have been ironed out.

Posted Monday, March 10, 2003

xDataBackupManager Pro v1.4

SealSoft has released xDataBackup Manager Pro v1.4. Changes include: Some improvements in the template; Manual backup function; Restore function; Logging function; Two new formats of dates for archive file name; Generation of unique file name (for compatibility with 8.3 names); Selective delimiter for archive file name; Integration with the new version xAccessManager Lite v1.4 (This version will be available soon); Bug fixes. Registered Users: Install kit password was changed and will be sent to you via email. Similar changes have been made in xDataBackup Manager Lite - this version will be available soon.

Posted Monday, March 10, 2003

EasyResizeAndSplit 1.04

EasyResizeAndSplit ver 1.04 is now available. Changes include: New split control type - image control; Support for Easy3DStyle; Open() method; New EVENT:SplitMove; Small modifications in the template; Redrawing now more optimized; Fixed some strange behavior of split controls.

Posted Monday, March 10, 2003

xReplacer (TX* Manager) For Clarion

SealSoft's xReplacer (TX* Manager) now works with TXD files (hence the name change from TXA Manager to TX* Manager). New in this release: Use Alt-Up and Alt-Down for quick

Goto to FilesHeader in TXDs; Some menu items and pop-up tips have changed; Button List moved from Action Menu to Goto Menu and now named "Goto Button...".; New "Close File" function added in File Menu; Generate comments column feature; Set marker in the text (save position) and go to marker; Use external editor to edit files; Downgrade Application, i.e. C5.5 TXA file to C5 TXA file; Option to reload file if changed on disk. New demo available.
Posted Monday, March 10, 2003

gREGPlus v4.51 Released

Gitano Software has released gREGPlus version 4.51. Until March 15, 2003 you can upgrade to gREGPlus from a competing product and save $100.
Posted Monday, March 10, 2003

Clarioners Database

Fill in a form and put yourself in SealSoft's Clarioneers database.
Posted Monday, March 10, 2003

xReplacer (TXA Manager) 1.0 Beta Build #003

New in this version of SealSoft's xReplacer: Version history list was added; Go to previous and next structure (Window, Report or Application) using Alt-Up and Alt-Down key; Edit any line of text in listbox (EIP); EnterKey in listbox opens EIP, Ctrl-EnterKey opens editor if you select embed code in listbox; Improvements to replacing algorithm; Bug in loading list of embeds fixed; New options in program preferences. New demo available.
Posted Monday, March 10, 2003

Newsletter Service Update

Sign up for one month and receive a second month free. Sign up for one year and receive 15 months. This offer is good until March 31 2003. You can pay month by month, or pre-pay for a full year in advance.
Posted Monday, March 10, 2003

ClarionPost.com/ClarionKB.com

Kelvin Chua's www.clarionpost.com and www.clarionkb.com are up and running now. Please remember that any links that are not Clarion-related nor beneficial to the Clarion community will be removed without any notice at all. All links to pornography or gambling sites will be removed without any notice.
Posted Monday, March 10, 2003

Reader Comments

Add a comment

Clarion Magazine

The Clarion Magazine GPF Challenge

Published 2003-03-14

This Clarion challenge is a bit different from those Clarion Magazine has run in the past. Normally, we'd be asking you to write some code that does something truly wonderful, probably using the minimum source code, and with an absolute excess of elegance and speed. But that's not what this challenge is about.

It's about GPFs. Tricky, nasty, hard to find GPFs. We want you to make them. And Russ Eggen, champion of the Clarion debugger, is going to track them down. See Russ's article on post mortem debugging in this issue.

To participate, you need to provide a complete application, which can be either an APP or a PRJ. Be sure to include all necessary source code, dictionaries, TPS files, whatever it takes to compile and run the application. And of course include some instructions on creating the GPF. You can be as tricky as you like in causing the GPF, as long as whatever you do involves the source.

Zip up your challenge and send it to editor@clarionmag.com. We'll award a six month subscription/renewal to the winner. Entries will be judged on the difficulty in finding and fixing the GPF, and the instruction value of the problem and solution (that is, you won't get as many points for coming up with a scenario that is virtually impossible to achieve under normal development conditions (is there such a thing?).

Reader Comments

Add a comment

CWPlus. Super-Tool! Easy Data Handling in Clarion IDE.
Download Now! You won't regret it!
IngasoftPlus

A Naïve Look At Pre-Emption

by Steven Parker

Published 2003-03-20

When I began writing about Clarion, the big concern in the community was getting Clarion Database Developer (the DOS predecessor to today’s Clarion) to work (sometimes, at all). And before you start with the comments on the "Clarion 3.x fiasco," you need to be aware that Clarion for Windows is built on the design innovations (if not the actual code) established in CDD. Among the more obvious carry-overs are replaceable drivers, the embed system, the formula editor (every bit as useful now as it was then) and generic, open templates. These are all CDD innovations.

My first articles concerned getting lookups to work the way I thought they should. Now, the Clarion community is concerned with COM objects and pre-emptive threads. Quite an evolution indeed.

("Real" programmers or persons able to comprehend Microsoft’s API "documentation" need read no further.)

The trouble with threads

Pre-emptive threads present not only a new set of challenges but a whole new level of challenge. The primary new challenge is that coding styles need to change. "Need," not "should" or "ought." These changes are mandatory.

Oh, okay, they’re not *really* "mandatory." If you will be creating SDI apps or expect to use the backward "compatibility" mode planned for Clarion 6, pre-emptive threads are a non-issue.

For the rest of us, global abuse has just got to be dealt with.

I am as guilty as anyone of using global data for inter-procedure communication and "flag" files for inter-process communication. In Clarion Professional Developer (CPD), there was little choice. In CDD, eventually, there *was* a choice. In fact, I rewrote those templates to allow full parameter passing; every procedure type supported parameters, received and passed/passed on (in some templates, I provided prompts for parameters received and parameters to pass on to the next procedure).

Luckily, I never feel into the trap that some did of using globals to store intermediate values, for temporary, intra-procedural storage. And, so far at least, I can't find a case where I use a global as a parameter-substitute and the called procedure is `Start`ed. (I'm not saying none exist, only that I haven't found them … yet.)

The fact that no STARTed procedure receives a parameter-substitute is very important. Because the two (or three) procedures using the variables are on the same thread, I can add the `Thread` attribute to those variables. With `Thread`ed variables, only procedures on that thread can see the values in those variables. This is SoftVelocity's number two recommendation for "upgrading" to the pre-emptive threading model.

Why is the `THREAD` attribute so highly recommended? What about other instances of the procedures? Think about this. Either other instances are on another thread or there are no other instances (can't have two instances on the same thread, can I?). The *only* way to get additional instances is to `Start()` them. That means additional procedures, if any, have their own copies of the variables. (Besides, as a matter of course, I allow only one instance of most procedures; I use Jim De Fabia's Thread Limiter Template. This eliminates the issue entirely, for me.)

So far I've been considering a single chain of procedures that use globals for parameters. What about multiple calling chains that use the same globals ("hey, I've already got this global variable, why not just use *it*? -- yeah, I've done that once or twice).

Well, as long as each chain is on a separate thread, no problem.

Okay. But now that C6 (finally!) provides a parameter prompt for a browse's update procedure, shouldn't I just change to doing it the right way? The "right way" guarantees everything will be copasetic. This is SoftVelocity's number one recommendation for upgrading to the pre-emptive threading model.

Consider a project I'm working on right now. Let's see, three EXEs, 14 DLLs … on second thought, I don't think so.

I understand the basic concept of pre-emptive threads. SoftVelocity has a white paper describing the changes (Thread Model Changes) if you're not up to speed on this subject. The real problem with pre-emptive threads is that while I understand the theory, I do not understand the techniques necessary to protect myself.

Specifically, I can't quite get my head around the defensive programming side of pre-emptive threads. And you know that Murphy's Law (and possibly several of its corollaries) will absolutely apply if I'm not prepared.

Newspeak, again

Microsoft, of course, is its usual helpful self in assisting us in understanding what is required. They have yet again invented a new (and entirely unnecessary and incomprehensible) vocabulary.

To control access to static (non-threaded and non-local) objects and data, Microsoft gives us "synchronization objects." Clarion 6 builds in support for three of them via several Interfaces: Critical Sections (IcriticalSection), Semaphores (ISemaphore), and Mutexes (IMutex).

Problems start with the fact that "synchronize" doesn't mean what it means in the rest of reality. As David Harms and Carl Barnes point out:

> The word "synchronize" really means to cause events to happen at the same rate or the same time. Synchronization objects like semaphores can be used this way, but more commonly synchronization objects are actually used to serialize access to shared resources so that no two threads use the resource at once.

"Serialize?" In other words, "synchronize" means something more like "restrict access by others." Often, if not usually, this "restriction" is "with extreme prejudice," i.e. complete blocking of access by other threads. But some of these "synchronizers" allow a definable number of other threads to access the data simultaneously.

So, others are blocked … except when you decide it's okay to risk your global data. Gee, thanks Mr. Gates.

Right.

To be charitable, the data read will be synchronized, i.e. the same. By making sure that no other procedure can write to my global queue or variable while I am reading from it, I am sure that I will get the data I expect to (see the example in SoftVelocity's white paper). Thus, all read data is synchronized in the standard sense of that word. (In Philosopher school, this was called extending the meaning of a word by metaphor. And some metaphors are just better than others.)

Next, we are introduced to the concept of "wait" methods (synchronization objects have a "wait" method and some have a "trywait" method; the latter are known as "waitable" – don't ask, neology is not an exact science). "Wait" is just another Microsoftism.

If you look at SoftVelocity's white paper on handling pre-emptive threads (Multi-Threaded Programming), you will see code like this:

```
IF THREAD() <> 1
  QLock.Wait()
```

"Wait" seems to mean "hang around (wait) until you can "get" the synchronization object. "Get" ="own, have possession." "Wait" ="the line forms in the rear, pal." But, if you don't have to "wait," you have access to whatever it is that you need access to (you were first in line).

Not waiting is good. At last, a similarity to the real world!

This may sound a bit amusing. But as David and Carl point out in part 2 of "Demystifying C6 Threading," this could mean waiting forever. And you might not even know about it.

A Case for the Defense

Waiting forever is bad. It is very bad. Creating a situation in which an infinite wait could happen is a very bad idea and … not recommended. To resurrect a term from the distant past, this is not very user friendly.

Consider this code (taken from a DOS project I am migrating to Windows):

```
DO UnitSchemefromPrice
GBL:UnitScheme = LOC:UnitPriceScheme
  CostWarningScreen
LOC:UnitPriceScheme = GBL:UnitScheme
DO UnitPricefromScheme
```

Markup/margin is calculated from unit price. The result is stored in local variable. It is then copied to a global parameter-substitute. Then `CostWarningScreen` is called.

`CostWarningScreen` is a procedure where the user can adjust unit price or margin/markup. (This entire sequence is triggered by a cost change during receipt of merchandise, by the way.)

When the user is done, the margin/markup is stored locally and unit price recalculated.

Now suppose that `GBL:UnitScheme` is used by another thread as a global parameter-substitute, say generating purchase order recommendations from inventory.

Further suppose that I foolishly did not thread `GBL:UnitScheme`. Suppose that I, even more foolishly, elected not to change the prototype of `CostWarningScreen` to something like this:

```
CostWarningScreen(*Decimal varName)
```

(Passing the variable as a parameter would also eliminate the need to read or write the value to the local variable. And, if every global variable were replaced by a procedure parameter, my need for global variables would be entirely moot, along with the rest of this article)

Assuming I don't pass the data as a parameter, however, I still need the global data. And if I haven't threaded the variable, then to protect `GBL:UnitScheme` from being changed by two procedures running at the same time, I obviously have to wrap reading and writing `GBL:UnitScheme` in a synchronization object.

Case 1: Suppose I start the order generating process and it takes 20 minutes to complete.

If I then try to start receiving merchandise (the other procedure using `GBL:UnitScheme`) and have a cost change, I won't get control of the synchronization object. So receiving will "wait." It will wait for 20 minutes or thereabout, until ordering completes.

Case 2: Now suppose I start receiving merchandise and get a cost change. Now the `CostWarningScreen` procedure is called. If I try to start ordering in the background, it will "wait" for me to complete the `CostWarningScreen` procedure. But, suppose I go to lunch instead of completing the form.

Now everything just sits there "waiting." And I'm none too pleased when I return from lunch, expecting ordering to be done and it isn't.

Case 3: This is Case 2 but I don't go to lunch. I complete my cost change. Ordering starts. If another cost change triggers `CostWarningScreen`, I'm hung (or appear to be).

Thinking through these cases, a couple of things are obvious.

(1) a "Wait" without a time limit has to be thought through very, very carefully

and

(2) this code is obviously a good candidate for redesign (synchronization objects seem utterly

inappropriate).

No wonder the docs state "Proper programming techniques are essential here to avoiding a ‘deadly embrace’, or deadlock" and encourage keeping control of a synchronization object for the *shortest possible amount of time*. In short, failure to think in terms of what can happen to a user in fractions of a second can have significant repercussions.

Next time: a synchronization object I actually understand (and can use!).

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.

Reader Comments

Add a comment

>Besides, as a matter of course, I allow only one instance...

Clarion Magazine

PostgreSQL 101: Security Basics

by David Harms

Published 2003-03-20

In the previous article in this series I discussed table and index creation in PostgreSQL, as well as some of the peculiarities of sequences, which are PostgreSQL's version of auto-incrementing values. I had intended to continue on to the psqlODBC driver in this article, but as it turns out there are a few more configuration and security issues I need to get out of the way first.

I said at the start of this series that this would be a sort of diary of my experiences. And this week I was reminded again that I find PostgreSQL's security measures a bit obtuse at times.

One caveat: I'm still using PostgreSQL 7.2, and there have been some security improvements in 7.3. I'll do my best to touch on those as I go.

In Part 1 I described some of the troubles I had connecting with the native Windows beta on my development machine. In Part 2 I briefly showed how to set up a user and database on a Linux box, and connect using the psql utility running in a terminal window, on that Linux box. This time around, however, I'll be using PostgreSQL on a Linux box over a LAN, eventually via the psqlODBC driver.

This shift from connecting via a local application to a network connection is important, because PostgreSQL is, by default, set to only accept local connections. For instance, the Linux server I have running on my LAN is sporting a brand new RedHat 8 install, including PostgreSQL. If I telnet (secure shell, actually) to the Linux box (which doesn't have its own monitor at the moment) and run psql, that application talks to the PostgreSQL postmaster program using a socket connection. It does not, however, use TCP/IP, and is not, in fact, configured to do so. That's important because any Clarion application will be using TCP/IP rather than a socket.

You can verify that PostgreSQL is not listening on the TCP/IP port (by default port 5432) by executing the following on the Linux server:

```
# telnet localhost 5432
Trying 127.0.0.1...
telnet: connect to address 127.0.0.1: Connection refused
```

If you see that message, then the first thing you'll need to do is tell PostgreSQL to start listening on the TCP/IP port. Locate the file `postgres.conf` on your system; on my Linux box it's in the `/var/lib/pgsql/data/` directory. Here's a partial listing of that file:

```
# PostgreSQL configuration file
# -----------------------------
#
# This file consists of lines of the form
#
#   name = value
#
# (The `=' is optional.) White space is collapsed, comments are
# introduced by `#' anywhere on a line.  The complete list of option
# names and allowed values can be found in the PostgreSQL
# documentation.  The commented-out settings shown in this file
# represent the default values.
# Any option can also be given as a command line switch to the
# postmaster, e.g., 'postmaster -c log_connections=on'. Some options
# can be changed at run-time with the 'SET' SQL command.
#==============================================================
#
#       Connection Parameters
#
#tcpip_socket = false
#ssl = false

#max_connections = 32

#port = 5432
#hostname_lookup = false
#show_source_port = false

#unix_socket_directory = ''
#unix_socket_group = ''
#unix_socket_permissions = 0777

#virtual_host = ''

#krb_server_keyfile = ''
...
```

As noted in the header, the commented out lines represent the default values, and `tcpip_socket` is set to `false`. Uncomment that line and change it to true (my thanks to

Jeff Slarve for pointing this out):

```
tcpip_socket = true
```

Save and close the file. You will need to restart PostgreSQL to make the change take effect – on my Linux box I use this command:

```
/etc/rc.d/init.d/postgresql restart
```

Try the telnet test again. It still doesn't work? Ah, it turns out there's one other thing you need to do, and that's set the appropriate permissions. Locate the `pg_hba.conf` file, in the same directory as postgres.conf. At the end of that file you will see something like the following:

```
# Put your actual configuration here
# ==================================
#
# This default configuration allows any local user to connect with any
# PostgreSQL username, over either UNIX domain sockets or IP.
#
# If you want to allow non-local connections, you will need to add more
# "host" records. Also, remember IP connections are only enabled if you
# start the postmaster with the -i option.
#
# CAUTION: if you are on a multiple-user machine, the default
# configuration is probably too liberal for you. Change it to use
# something other than "trust" authentication.
#
# TYPE DATABASE IP_ADDRESS MASK            AUTH_TYPE  AUTH_ARGUMENT

#local all                                 trust
#host  all      127.0.0.1  255.255.255.255 trust

# Using sockets credentials for improved security. Not available everywhere,
# but works on Linux, *BSD (and probably some others)

local   all     ident   sameuser
```

These are all comments except for the last line, which basically says that anyone logged on to the machine as a local user can access any database. Oh, and this document does also point out that you can use the `–i` option when starting the postmaster to enable TCP/IP connections, but it's more likely that you will have a startup script for PostgreSQL, in which case you should use postgres.conf setting. There's a lot more to `pg_hba.conf` than I've shown here – it really does give a lot of useful (if a bit cryptic) information on security settings.

In order to connect to PostgreSQL via TCP/IP, you must something like the following to pg_hba.conf:

```
#TYPE  DATABASE IP_ADDRESS MASK            AUTH_TYPE  AUTH_ARGUMENT
host   all      127.0.0.1  255.255.255.255 trust
```

This line specifies the host access that is allowed. As written, it permits connection to all databases from the localhost IP address only, and does not require any authentication. If you can connect from localhost, you're good to go. In a way this is a fairly secure approach, assuming you have complete control over all applications executing on that machine. It does *not* allow an application to connect to the database from any other machine. Well, sort of.

Last year I wrote an article about SSH tunneling, which is a great way to encrypt remote database connections. You could set up an SSH tunnel from a Windows box to the Linux server and connect to the PostgreSQL database using the above configuration. The SSH server is a local application, so it can talk to PostgreSQL, and your SSH client (on, say, a Windows machine) talks to the SSH server.

In any case, you'll probably want at least some level of authentication. In order to use authentication you'll need to set passwords for your users. In Part 1 I showed how to use `createuser` to create PostgreSQL users. With the `-P` parameter, you can get createuser to prompt for a password:

```
$ createuser -P cmag
Enter password for user "cmag":
Enter it again:
Shall the new user be allowed to create databases? (y/n) n
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
```

You can also create and modify users from within the `psql` utility. Typically you'll need to sign on as the `postgres` superuser. The easiest way is to log in to the Linux box as root and then `su` to the `postgres` user:

```
$su root
Password:
# su postgres
$ psql template1
Welcome to psql, the PostgreSQL interactive terminal.

Type:  \copyright for distribution terms
       \h for help with SQL commands
       \? for help on internal slash commands
       \g or terminate with semicolon to execute query
       \q to quit

template1=#
```

Note that you must supply a database name when connecting, as `psql` defaults the database name to the user name and there is (probably) no database with the name `postgres`. There is always a `template1` database, although it's not likely you'll ever modify it since it is, as the name indicates, a template for databases you create. Once logged in you can use the `CREATE`

`USER` and `ALTER USER` statements to modify user settings.

> **WARNING**: The `CREATE USER` statement, by default, creates a superuser, that is someone who can create databases *and* other users. So unless you're prepared to use the full syntax of `CREATE USER` you're probably better off with the command line `createuser` utility.

Now that you have a user with a password, you'll probably want a database with a matching name (or vice versa). On the face of it this seems like an odd requirement, and in fact there are ways around it. But one of the authentication options in the `pg_hba.conf` file is `sameuser`, which tells PostgreSQL to give access to a database only when the user name and the database name match. Here's a line for `pg_hba.conf` that restricts users to same-named databases on the local 192.168 network:

```
host sameuser 192.168.0.0 255.255.0.0 password
```

Creating a database for the `cmag` user is easy:

```
$ createdb cmag
CREATE DATABASE
```

It is possible to specify a list of users for a given database by setting up a separate file with user names and, optionally, passwords. But PostgreSQL is not nearly as configurable for security as its rival MySQL, where all permissions are table-based. Fortunately, PostgreSQL 7.3 adds some improvements – you can specify multiple databases on one line in `pg_hba.conf`, and you can also specify multiple users or user groups at the end of the line as `AUTH_ARGUMENT` data, or so I've read.

If you're stuck with 7.2, then have a look in the PostgreSQL docs for the `pg_passwd` utility, which you can use to maintain password files for individual databases.

Summary

PostgreSQL's authentication system can be a bit quirky, and you can reasily run afoul of the default settings when you're trying to configure the database for ODBC access. You need to ensure that PostgreSQL is set up to listen on its TCP/IP port (by default 5432) and you also have to configure the `pg_hba.conf` file to allow specific remote connections. Once you have those two things sorted out you're ready to start playing with data using Clarion, right? Well, almost. There are still a few minor issues about table and database permissions to get sorted out. More on that next time.

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

Reader Comments

Add a comment

Clarion Magazine

For marketing your Applications
and Developer Accessories or to
purchase other 3rd Party Tools . . .
Developer PLUS tm

Ads · Comments · Writers! · Privacy · Contacts · PDFs · Freebies · Open Source

Topics > Tips/Techniques > Clarion Language

My First Function Library

by A D TELFORD

Published 2003-03-21

When I go to the local toy store I notice a large number of toys all vying for my attention. My First Train Set, My First CD Player, My First Cell Phone … The very words *My First* tells me that if I buy this toy, I will have so much fun and will remember it for life. In the same way I hope to entice you, the Clarion programmer, on how to make your *first* function library. And yes, it will be important to you! You will enjoy it! You will remember it for life! Convinced yet? You will be. Let me give you an example.

I have noticed several people complain in the newsgroups about the lack of a *maximum* or *minimum* function. The comments usually go something like, "Every other language has this feature – why not Clarion?" And of course, they're right – Clarion should have these features. But I don't have to wait for SoftVelocity. If I have my own library, I can add this function in a matter of minutes, and then use it everywhere. Hmmm. I think I've heard this before – *write once, use everywhere.*

First I need to write these new functions, which will return the maximum or minimum of two passed values, of any simple type:

```
Max   Procedure(? p1, ? p2),?
  code
  return( choose( p1 >= p2, p1, p2))

Min   Procedure(? p1, ? p2),?
  code
  return( choose( p1 <= p2, p1, p2))
```

> **Note**: The question mark (`?`) is an untyped value-parameter, which allows the single procedure to work for `STRING`, `LONG`, `DECIMAL`, `REAL`, and other data types. Refer to the Clarion Help on **Prototype Parameter Lists**.

I can almost hear you say, "But that doesn't save much time. It's just as easy to code the `choose` statement directly." But let me ask you, which of the following lines is easier to read, `min` or `choose`?

```
result = min( figure1, figure2)
result = choose( figure1 <= figure2, figure1, figure2)
```

What if I replace `figure1` and `figure2` with a complex calculation? Which one is easier to read now?

```
result = min( (figure1 * 8/100 + 0.25), (figure2 * 6/100 + 1.20))
result = choose( (figure1 * 8/100 + 0.25) <= (figure2 * 6/100 + 1.20),|
  (figure1 * 8/100 + 0.25), (figure2 * 6/100 + 1.20))
```

Now I'm really saving time. `MIN` is much easier to read, and if I later change the calculation used, I only have to change it in one place instead of two.

And that's not all. If `MIN` or `MAX` was a built in Clarion function, then I would be limited to using it exactly as specified in the Language Reference manual. Since I'm designing the function, I can modify it at my pleasure. What if I'm not content with passing only two parameters? Sometimes I'd like to find the smallest of three values, or maybe even four or five. Clarion can handle this with ease:

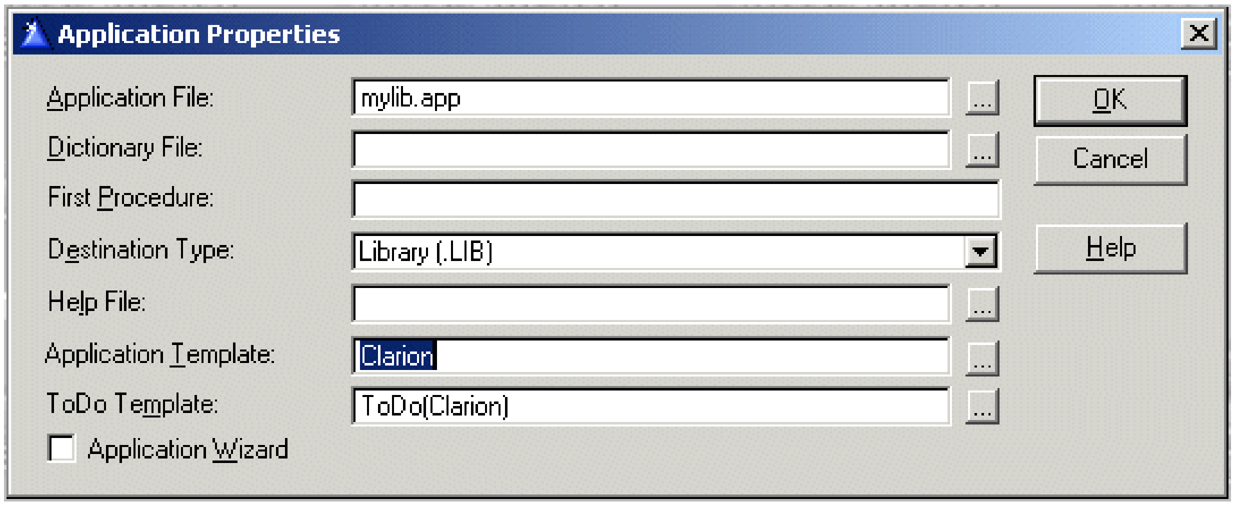
```
Min   Procedure (? p1, ? p2, <? p3>, <? p4>, <? p5>),?
ret   Any,auto
  CODE
  ! return the minimum value from a list of 2-5 items
  ret = p1
  if p2 < ret then ret = p2.
  if ~omitted(3)
    if p3 < ret then ret = p3.
  end
  if ~omitted(4)
    if p4 < ret then ret = p4.
  end
  if ~omitted(5)
    if p5 < ret then ret = p5.
  end
  return(ret)
```

Now we're talking! I don't know about this name though. `MIN` seems so boring. What about calling the function something like `smallest`, `minimum`, `lowest`, `least`, or even `tiniest?` Obviously some names work a lot better than others. But since Clarion didn't create this function, I can call it anything I like.

Now that I have extended the Clarion language and added the new `MIN` and `MAX` statements, where do I put these procedures, and how do I use them?

Creating My Library

The beauty of a function library is that I can store all my functions inside an ordinary APP file, which will then be compiled to a LIB file, and made available for use everywhere. First I create a new application and call it **mylib.app**, filling out the fields as shown in (figure 1), and then press **OK**.

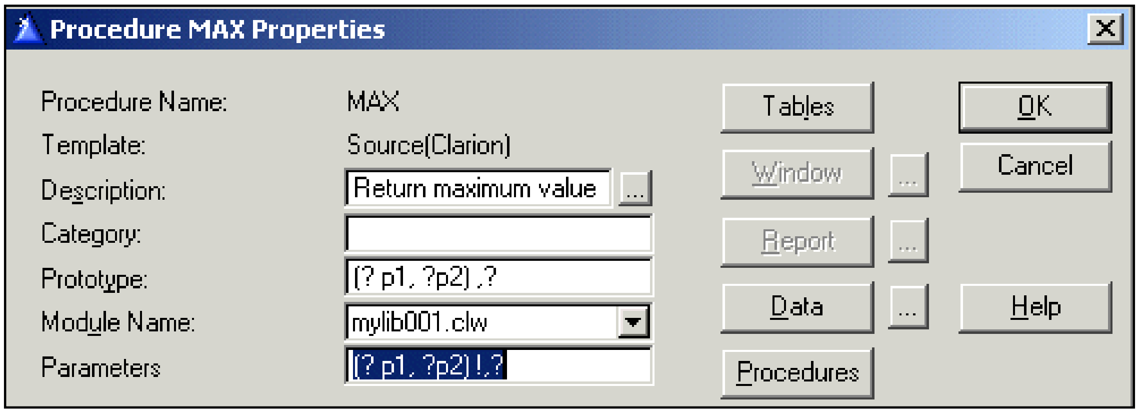


Application File:	mylib.app
Dictionary File:	
First Procedure:	
Destination Type:	Library (.LIB)
Help File:	
Application Template:	Clarion
ToDo Template:	ToDo(Clarion)

Figure 1. Creating the function library application

Notice that no dictionary is required (you may need to go to **Setup|Application Options** in the IDE and uncheck **Require a dictionary**). I want these functions available everywhere, and not tied to any specific app. There is no first procedure to run, as the destination type is a LIB and not an EXE. Surprisingly, I find it works best when I choose the **Clarion** template chain. The **ABC** template chain automatically adds the ASCII database driver to every app, even when it's not required, which can cause errors later.(Note – if you don't have **Clarion** available as a choice for template type, then you will need to register the Clarion template chain. The name of the file to register is **cw.tpl**)

Now, I can add my new `MIN` or `MAX` function. I press **Insert** to add a new procedure, give it the name **MAX**, and make sure the template type is **Source.** One little trick I use here, is to enter the full **Prototype** including return value. I then copy this value, paste it into the **Parameters** field, and enter an exclamation mark (!) before the return value (figure 2). This comments the return type in the generated source code, but still allows it to compile.



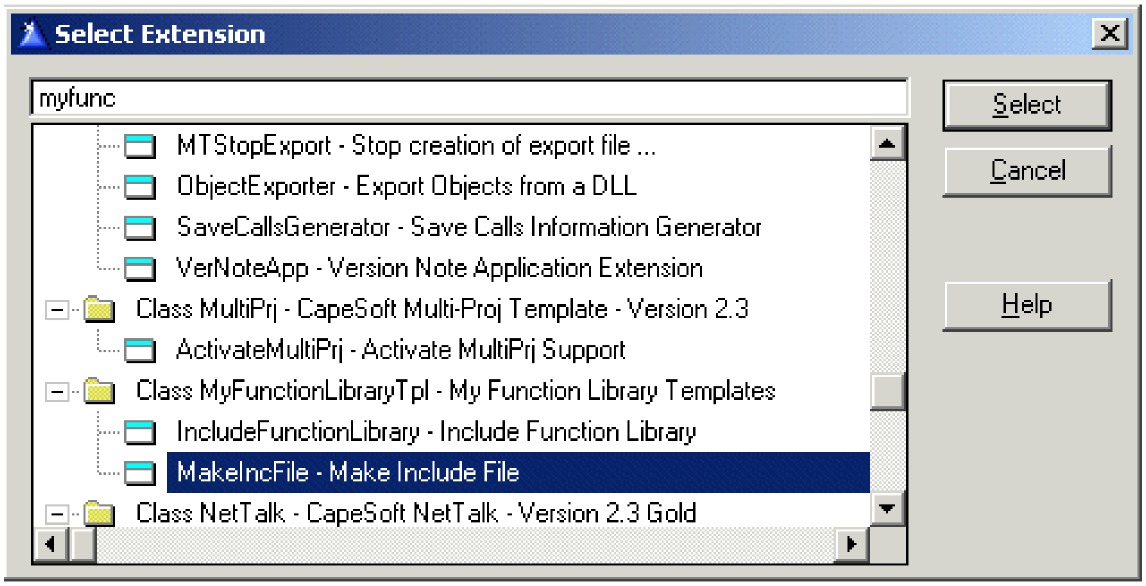
Procedure MAX Properties

Procedure Name:	MAX
Template:	Source(Clarion)
Description:	Return maximum value
Category:	
Prototype:	(? p1, ?p2) ,?
Module Name:	mylib001.clw
Parameters	(? p1, ?p2) !,?

Figure 2

I now enter the embed code for the MAX procedure, and then compile the app.

Now, if this was a built-in Clarion function, it would be instantly available for use everywhere else. The Clarion compiler can identify these functions because they're listed in **builtins.clw** inside the Clarion Libsrc folder. For me to use these function inside another app, I must create a similar file (which I've called **mylib.inc**), listing the names and prototypes of each function. This is a great job for a global extension template (see Figure 3) which you can download at the end of the article (Register the myfunlib.tpl template). Every time I add a new procedure and compile, the template updates the MYLIB.INC file with the new procedure prototypes, and so makes maintenance a real no-brainer.



Select Extension

myfunc

```
    MTStopExport - Stop creation of export file ...
    ObjectExporter - Export Objects from a DLL
    SaveCallsGenerator - Save Calls Information Generator
    VerNoteApp - Version Note Application Extension
Class MultiPrj - CapeSoft Multi-Proj Template - Version 2.3
    ActivateMultiPrj - Activate MultiPrj Support
Class MyFunctionLibraryTpl - My Function Library Templates
    IncludeFunctionLibrary - Include Function Library
    MakeIncFile - Make Include File
Class NetTalk - CapeSoft NetTalk - Version 2.3 Gold
```

Figure 3. Choosing the MakeIncFile global extension

How do I use my new function library?

Now that I've created my function library (mylib.lib) complete with include file (mylib.inc), it takes less than a minute to make it available for use inside any another Clarion application, and no further templates are required. From the **Application** pulldown, select **Insert module**, and specify **ExternalLib** (see figure 4). All that's needed is for me to enter the name of my library and its associated include file (Figure 5).

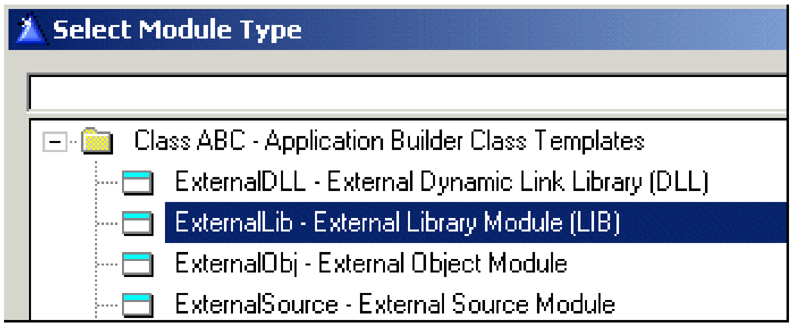
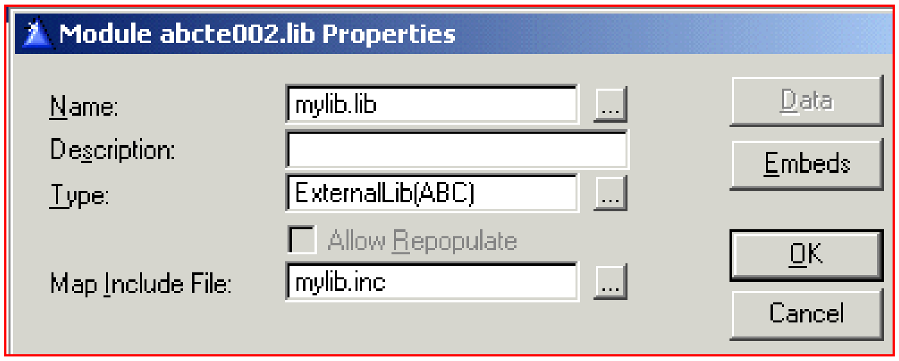

Select Module Type
Class ABC - Application Builder Class Templates
ExternalDLL - External Dynamic Link Library (DLL)
ExternalLib - External Library Module (LIB)
ExternalObj - External Object Module
ExternalSource - External Source Module

Figure 4. Selecting the ExternalLib module


Module abcte002.lib Properties

Name: mylib.lib
Description:
Type: ExternalLib(ABC)
Allow Repopulate
Map Include File: mylib.inc

Figure 5. The library module properties

I can now use my new `MAX` and `MIN` functions from inside any embed point. And now that the structure is set up, I can easily add to this function library as I think of new ideas. As an exercise the reader may like to take the `ExportQtoCsv` function that I described in my last article on Debugging Queues with Excel, and add it to your function library.

Summary

In this article I've shown how you can create a function library. I gave an example by extending the Clarion language, and adding the `MAX`/`MIN` functions. As the creator of these new functions, I can even choose what to name these functions, and what limits they have.

I then showed how to store all these useful functions inside a regular Clarion app, and then with an aid of an extension template, have an include file automatically generated. Then with a few seconds typing I showed how I can add my external library module into any other Clarion app. Once the structure is set-up, I can easily add to my library as I think of more ideas.

An OOP Class library may be more suitable where you have a lot of related functions which need associated data. But for straight functions (like `MAX` or `MIN`) then a library is more appropriate, and also simpler for beginners, as the functions can be easily stored inside an APP rather than requiring a hand-coded module.

Everybody needs new tools in their tool kit (or toys in their toy box). I hope you like the idea of a function library. Next time you get a good time-saving idea, why not see if it's suitable to add to *My First Function Library*.

Download the source

Alan Telford has been programming in Clarion since 1994. He is the Chief Software Developer at Maxtel Software Ltd, a New Zealand software company specializing in writing back office computer solutions for McDonald's Family Restaurants and other similar markets.

Reader Comments

Add a comment

Alan, ABC adds the ASCII driver due to the ErrorClass. ...
Alan, Great article. Learning to do a function...
A nice article. Writing library functions in an App...

Clarion Magazine

Data Structures And Algorithms Part XVII - Decompression

by Alison Neal

Published 2003-03-27

In my last article I discussed Huffman's compression algorithm, which provides about 40% compression for text files. In this article I will show how to decompress the now compressed file.

The process of decompressing a file is similar to compressing a file, and requires the following steps:

1. Read the compressed file header to build a frequency array of how many times each character occurs in the file.
2. Build a Priority Queue and Huffman Tree from the frequency array.
3. Read the remainder of the compressed file, decoding the binary codes in the file and writing the associated ASCII character to the decompressed file.

Step 1. Read the compressed file header

The header of the compressed file contains the output frequency array, created in the compression algorithm. Reading the frequency array back is very similar to the way in which it was originally output:

```
Read:Freq            ROUTINE
  DATA
jLower  BYTE(0)
jUpper  BYTE(0)
ch      BYTE(0)
CntByte BYTE(0)
LenCode BYTE(0)
l       BYTE(0)
u       USHORT(0)
f       ULONG(0)
```

```
j        LONG
j0       LONG
k        BYTE(0)
 CODE
  Cancel = ios_imp.SetPointer(0,0)
  LOOP j = 1 TO MAXIMUM(dat:freq,1)
    dat:freq[j] = 0
  END
  Cancel = ios_imp.ReadFile(ADDRESS(jLower),1)
  IF ~Cancel THEN Cancel = ios_imp.ReadFile(ADDRESS(jUpper),1).
  j = jLower
  LOOP WHILE j <= jUpper
    IF Cancel THEN BREAK.
    j0 = j
    IF ios_imp.ReadFile(ADDRESS(CntByte),1) THEN BREAK.
    LenCode = BSHIFT(CntByte,-6)
    l = cntByte - BSHIFT(LenCode,6)
    j = j0 + l
    LOOP WHILE j0 < j
      IF Cancel THEN BREAK.
      CASE LenCode
        OF 0
          f = 0
        OF 1
          Cancel = ios_imp.ReadFile(ADDRESS(ch),SIZE(ch))
          f = ch
        OF 2
          Cancel = ios_imp.ReadFile(ADDRESS(u),SIZE(u))
          f = u
        OF 3
          Cancel = ios_imp.ReadFile(ADDRESS(f),SIZE(f))
      END !CASE
      dat:freq[j0] = f
      j0+=1
    END !LOOP WHILE
  END !LOOP j
```

The first values to be read from the compressed file are the lowest and highest populated values of the frequency array. In the example from my last article: *"Steve sells sea shells by the sea shore"*, the frequency array was populated thus:

Chr	**Val / Idx**	**Count**
[space]	32	7
S	83	1
'a'	97	2
'b'	98	1
'e'	101	8
'h'	104	3

'l'	108	4
'o'	111	1
'r'	114	1
's'	115	7
't'	116	2
'v'	118	1
'y'	121	1

Figure 1. The frequency array

The lowest index value being 32 and the highest index value 121. This provides the application with the "relevant" (rather than real) lower and upper bounds (range) of the frequency array. It tells the application that all elements before 32 are zero, and all elements after 121 are zero.

Next, the remaining positions (32 to 121) of the array, which are not zero, need to be populated with the correct frequency. At this point the program enters the loop and reads the first control byte, which contains the sequence format in the first two bits and the sequence length in the remaining six bits. In this example the control byte will be 01000001, meaning that the frequency about to be read is only one byte in length (7), and there is only one frequency in this sequence (32), this being the frequency for position 32 of 7.

The next loop has the purpose of reading in all the frequencies of the sequence indicated by the control byte, so in this example the second loop only iterates once with the reading of 7 into the frequency array for position 32.

The outer loop continues reading the next control byte, which in the example is 00110010. The first two bits (00), tell the application that the next sequence is all zeros. The next six bits tell the application how many zero positions there are in the frequency array, this number being 50. The second loop ensures that the next 50 positions of the frequency array are initialized to zero.

The outer loop iterates again and a third control byte is read. This time the control byte relates to position 83, which has a frequency of 1. The control byte therefore contains 0100001, the format of this sequence is 01 (requiring only 1 byte to read) and there is 000001 or only one frequency to read in this sequence.

Thus the procedure continues, reading in each control byte and their frequencies into the character frequency array until it is populated, as shown in Figure 1.

Step 2. Build the Priority Queue and Huffman Tree

The decompression code to build the Priority Queue and Huffman Tree from the frequency array is exactly the same code that is used in the compression algorithm - see that article. The Huffman Tree allows the application to decode the character codes as they are read from the compressed file. Figures 2 and 3 give the resultant Huffman Tree.

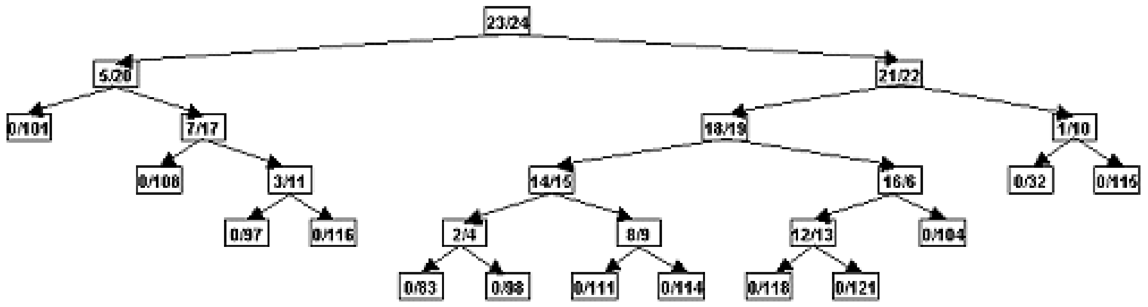


Figure 2. The Huffman tree (tree view)

Huffman Tree		
IDX	**Left**	**Right**
1	0	32
2	0	83
3	0	97
4	0	98
5	0	101
6	0	104
7	0	108
8	0	111
9	0	114
10	0	115
11	0	116
12	0	118
13	0	121
14	2	4
15	8	9
16	12	13

17	3	11
18	14	15
19	16	6
20	7	17
21	18	19
22	1	10
23	5	20
24	21	22
25	23	24

Figure 3. The Huffman tree (table view)

Step 3. Read the remainder of the file and decode.

The `Write:Decompress` routine reads the remaining compressed file and decodes it.

```
Write:Decompress    ROUTINE
 DATA
p1           LONG(0)
b            LONG(0)
nBits        LONG(0)
buf1         LONG(0)
buf2         LONG(0)
buf3         LONG(0)
xchar        STRING(1)
i            LONG(0)
tmp          BOOL(0)

  CODE
  p1 = Tr.GetRoot()
  Cancel = ios_imp.ReadFile(ADDRESS(buf1),1)
  IF ~Cancel
    Cancel = ios_imp.ReadFile(ADDRESS(buf2),1)

    LOOP
      IF Cancel THEN BREAK.
      tmp = ios_imp.ReadFile(ADDRESS(buf3),1)
      nBits = CHOOSE(tmp = FALSE, 8, buf2)
      LOOP i = nBits TO 1 BY -1
        b = BSHIFT(BSHIFT(buf1,31-(i-1)),-31)
        p1 = CHOOSE(b = 0,Tr.GetL(p1),Tr.GetR(p1))
        IF Tr.GetL(p1) = 0
          xChar = CHR(Tr.GetR(p1))
          Cancel = ios_exp.WriteFile(ADDRESS(xChar),SIZE(xChar))
          p1 = Tr.GetRoot()
        END
      END
      IF tmp = TRUE THEN BREAK.
      buf1 = buf2
      buf2 = buf3
    END
  END
```

The `huffTree` get methods are as below:

```
huffTree.GetRoot          PROCEDURE()
  CODE
  SELF.huffRoot = SELF.n - 1
  RETURN SELF.huffRoot

huffTree.GetL             PROCEDURE(LONG pIdx)
  CODE
  RETURN SELF.HuffL[pIdx]

huffTree.GetR             PROCEDURE(LONG pIdx)
  CODE
  RETURN SELF.HuffR[pIdx]
```

The data file is read one byte at a time and the tree is traversed, as each bit in the byte indicates. Taking the example from my last article, *"Steve sells sea shells by the sea shore"*, the following codes are applied:

Letter	Code
'e'	00
'l'	010
'a'	0110
't'	0111
'S'	10000
'b'	10001
'o'	10010
'r'	10011
'v'	10100
'y'	10101
'h'	1011

[space]	110
's'	111

This is how the compressed data breaks down:

- Byte 1: 10000011 (S and beginning of t)
- Byte 2: 10010100 (Rest of t, e and v)
- Byte 3: 00110111 (e, space and s)
- Byte 4: 00010010 (e, l and l)
- Byte 5: 11111011 (s, space and beginning of s)
- Byte 6: 10001101 (end of s, e, a and beginning of space)
- Byte 7: 10111101 (end of space, s and beginning of h)
- Byte 8: 10001001 (end of h, e, l, and beginning of second l)
- Byte 9: 01111101 (end of l, s, space, and beginning of b)
- Byte 10: 00011010 (end of b and beginning of y)
- Byte 11: 11100111 (end of y, space, and t)
- Byte 12: 10110011 (h, e, and beginning of space)
- Byte 13: 01110001 (end of space, s, e and beginning of a)
- Byte 14: 10110111 (end of a, space, and s)
- Byte 15: 10111001 (h and beginning of o)
- Byte 16: 01001100 (end of o, r and e)

Working through this example, `Buf1` reads in 10000011 (Byte 1) and `Buf2` reads in 10010100 (Byte 2). When the loop first iterates `Buf3` reads 00110111 (Byte 3). As the third read was successful the first buffer must contain 8 bits, and so `nBits` is made to equal 8. The nested loop moves through the first buffer bit by bit, traversing the tree dependent on the bit value.

I can decipher the codes contained in `Buf1` by using Figure 2. The first bit is a 1, which means a move to the right child. That places me on the node containing 21/22, which is not a leaf node as `HuffL` is not zero at this point. The second bit is a 0 (zero), which means a move to the left child (21), which contains 18/19. The next 0 (zero) bit means another move to the left (18), which is the node containing 14/15. The fourth bit is also a 0 (zero) requiring another move to the left (14), which is the node containing 2/4. The fifth bit is also a 0 (zero), so left again to the node (2) containing 0/83. As `HuffL` contains a zero value this must be a leaf node, and thus 83 must be a valid character valid, giving the capital 'S' to write to the uncompressed file.

So far the procedure has only read the first 5 bits of the 8 bit buffer, so now `p1` is reset to being the root node, so that the process can begin all over again with the next code. The sixth bit is a 0 (zero) which requires a move to the left child (23) which is the node containing 5/20. The seventh bit is a 1 which requires a move to the right (20) which is the node containing 7/17.

The eighth bit is also a 1, requiring another move to the right (17) which is the node containing 3/11.

At this stage a leaf has not been found, but all eight bits have been used; `buf1` is made to equal `buf2`, 10010100 (Byte 2), and `buf2` is made to equal `buf3`, 00110111 (Byte 3). Another byte, 00010010 (Byte 4), is then read into `buf3`.

As the last read did not fail `Buf1` must contain 8 bits of data, so the nested loop cycles from 8 to 1, analyzing each bit position. `P1` still refers to node 17, which contains 3/11, and the first bit in the new byte is a 1 requiring a move to the right hand child (11) containing 0/116. As `HuffL` equals 0, this must be a leaf node making `HuffR` a valid character value, being 't'.

The process continues decoding the bits of each byte read from the compressed data file. In the example: *"Steve sells sea shells by the sea shore"*, each byte is full. This may not always be the case however, which is why three buffer variables are used in reading the compressed file. In my previous article I mentioned that in writing the compressed codes, if the last byte is not full, then zeros are appended to the end of the variable, filling up the byte. Then the number of remaining bits is written to the file. Therefore if a read into buffer 3 fails (there is nothing more to read), the number of bytes relevant too the algorithm will be stored in buffer 2:

```
nBits = CHOOSE(tmp = FALSE, 8, buf2)
```

This makes sure that additional letters aren't added to the end of the decompressed file because of the extra zeros.

Summary

The Huffman compression algorithm is the same algorithm as has been used for pkzip, mpeg compression, and jpeg compression. It's a surprisingly simple idea, which has a myriad of uses.

In my next article I'll be looking into a new data structure known as the Graph. The Graph structure has an infinite number of uses, including allowing you to map many to many relationships in memory. The Graph has been applied to many applications, to name but a few these include travel booking, route planning and project planning.

Download the source

Alison Neal has been using Clarion since 2000, whilst working for Asset Information Systems (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information

Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarioNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, Manufacturers, Military & Government, Legal & Financial, and Retail.

Reader Comments

Add a comment

Hey Alison if you keep churning out articles like this I'm...

Clarion Magazine

Demystifying C6 Threading (Part 3)

by David Harms and Carl Barnes

Published 2003-03-28

Part one of this series introduced the concept of shared resources, and showed how Clarion 6's new threading model introduces the prospect of corrupted global data. Part two showed how to use some of Clarion 6's synchronization objects to manage shared resources *when necessary*. But there are still a couple of synchronization objects to discuss: the ReaderWriterLock, and the CriticalProcedure.

ReaderWriterLock

ReaderWriterLock is not a Windows synchronization object that you will find in the Win32 API or on MSDN (although there is an implementation class in .NET). SoftVelocity's version uses various objects to control reading and writing under rules that are still being documented (and for that matter, the code may still be evolving – there were significant changes between EA1 and EA3). At this time they are not releasing the source code. These are the base rules:

1. Only one thread may write at one time
2. No reading while writing, no writing while reading
3. Multiple readers may be active
4. ReaderWriter only works within a Windows process (i.e. not across multiple applications)

Most likely SoftVelocity's ReaderWriterLock uses a critical section and a semaphore. In the downloadable source you can find an example of a ReaderWriter Carl has adapted from the book Programming Applications for Microsoft Windows, by Jeffrey Richter.

In EA2 the ReaderWriter added a new option for writer priority (which is the default). This applies in the situation where a writer is waiting for readers to complete reading and release so the writer can get a writer lock. When writer priority is used any new readers trying to get a reader lock will be forced to wait for the writer to gain its lock and release.

With reader priority the new readers ignore waiting writers and are able to successfully wait and get a reader lock. This means a constant stream of readers can "filibuster" a writer into waiting forever. It would probably be an unusual situation for this to occur in a typical Clarion user-driven app. But in general if a writer has something new to write you probably want the readers to get that new info as soon as possible, and the new writer priority feature allows that to happen.

To use a ReaderWriterLock (or RWL for short) you must first declare it in the same scope as the shared resource you are trying to protect. So for global data you would declare it globally, and for protecting class data you would declare it inside the class. (This scope rule of thumb applies to all the thread synchronization objects.) Some time during program startup or class initialization you then need to "new" the interfaces. Below is an example of a Class using a RWL:

```
MyData      CLASS
Construct        PROCEDURE
Destruct        PROCEDURE
ReadWrite        &IReaderWriterLock
Reader        &IsyncObject
Writer        &ISyncObject
UserName        STRING(32),PRIVATE
SetUserName    PROCEDURE(STRING Name)
GetUserName    PROCEDURE(),STRING
        END
MyData.Construct    PROCEDURE
    CODE
    SELF.ReadWrite &= NewReaderWriterLock()
     ASSERT(~SELF.ReadWrite&=NULL)
    SELF.Reader &= SELF.ReadWrite.Reader()
    SELF.Writer &= SELF.ReadWrite.Writer ()
MyData. Destruct    PROCEDURE
    CODE
    IF ~SELF.ReadWrite &= NULL
         SELF.Reader.Kill()
         SELF.Writer.Kill()
         SELF.ReadWrite.Kill()
         SELF.ReadWrite &amp;= NULL
     END
```

This example class has one data property labeled `UserName` that is private to the class (so that it can be fully protected and encapsulated). To read and write this property the `Get...` and `Set...` methods below use the RWL to prevent data corruption:

```
MyData.SetUserName    PROCEDURE(STRING Name)
    CODE
    SELF.Writer.Wait()
    Self.UserName = Name
    SELF. Writer.Release()
MyData.GetUserName    PROCEDURE()
Name        LIKE(SELF.UserName),AUTO
    CODE
    SELF.Reader.Wait()
    Name = Self.UserName
    SELF.Reader.Release()
    RETURN Name
```

The ReaderWriterLock interface promises to be an important tool for managing global data in situations where there is a lot of contention for both reading and writing the data. It cannot be used for protecting queue operations as reading a queue causes writing to the RTL, and so multiple queue readers are not safe.

CriticalProcedure

There is one other synchronization tool in C6 that you may (or may not) find useful, and that is the `CriticalProcedure` class. `CriticalProcedure` is simply a class that wraps the wait() of a critical section and provides an automatic release(). (A better name for this class might be "safe wait" as its main feature is protecting you from forgetting to release before exiting a procedure or method.)

The `CriticalProcedure` class can be used with any object interface based on `IsyncObject`, which includes `ICriticalSection`, `ISemaphore` and `Imutex`, as well as the ReaderWriter's Reader and Writer interfaces. The complete source for this class is included with C6 so you can read the 20 or so lines to see exactly how it works.

Consider a situation where a block of code that enters (locks) a critical section must perform a number of tasks, any one of which causes a failure and exit of the procedure as a whole. You issue a critical section `wait()`, and begin running the code. If an error occurs and you're aborting the procedure, you must be sure to call `release()` on the critical section before you return from the procedure. If you don't call `release()`, then any other code that uses that critical section will be left waiting.

`CriticalProcedure` is essentially a class wrapper for a critical section, that uses an automatic destructor to ensure the critical section gets released. If you declare an instance of a class at the procedure level, the Clarion runtime will `NEW()` (allocate memory for) that class when the procedure loads, and `DESTROY()` the class when the procedure terminates. The `CriticalProcedure` class has an `Init` method which you must call, passing a critical section, and a `Destruct` method which is called automatically on class destruction, and which contains a call to the critical section's `Release()` method. There is also a `Kill()` method you can call if you want to release the critical section manually.

Here's some example code taken directly from the Clarion 6 INIManager class in ABUtil.CLW. This class uses a critical section to ensure that only a single thread updates the INI file. It uses a `CriticalProcedure` in each method that needs to access the INI file to insure that the critical section is always released when the method exits. The `CritSect` critical section object is created when the `INIClass` is instantiated.

```
INIClass        CLASS
critSect            &ICriticalSection,PRIVATE
...much more...
                END

INIClass.Construct procedure
  CODE
  SELF.critSect &= NewCriticalSection()

INIClass.Destruct procedure
  CODE
  SELF.critSect.kill()

INIClass.Update PROCEDURE(STRING Sec,STRING Name,STRING Value)
critProc  CriticalProcedure
  CODE
  critProc.init(SELF.critSect)
  SELF.Update(Sec, Name, Value, SELF.FileName)
```

In any method that accesses the INI file (e.g. the `Update` method code shown above) there is a `CriticalProcedure` declared inside that method. In the `critProc.init(SELF.critSect)` call the class code will perform a `Wait()` on the critical section to lock it. When the `Update` method exits, the automatic disposal of the `CriticalProcedure` object will call its `Destruct` method. The destructor performs a `Release()` on the critical section.

There are a few points to keep in mind about `CriticalProcedure`. Take a look at the source for the `Init` method:

```
CriticalProcedure.Init PROCEDURE (*ISyncObject O)
  CODE
  SELF.Kill()
  IF NOT O &= NULL
    O.Wait()
    SELF.Sync &= O
  END
  RETURN
```

Here's a *very important point*: If you do not create an actual critical section object, but just pass a null reference (which is what would happen if you slavishly followed the example in the EA2 docs) the critical procedure will run, but will not in fact use the critical section at all! This is an easy point to miss, because the `Init` method does not post any kind of error if you pass a null object. This is intentional – apparently the idea is you can decide whether or not to activate the critical procedure functionality by whether you pass an object or a null reference.

Here's the code for the `Destruct` and `Kill` methods:

```
CriticalProcedure.Destruct PROCEDURE()
  CODE
  SELF.Kill()
  RETURN

CriticalProcedure.Kill PROCEDURE()
  CODE
  IF NOT SELF.Sync &= NULL
    SELF.Sync.Release()
    SELF.Sync &= NULL
  END
  RETURN
```

The `Kill` method has a call to `Release()`, which allows the next waiting critical procedure that is using this `ISyncObject` to run. After releasing the `Sync` pointer is `NULL`ed to insure that if `Kill()` was called the destructor does not try to release the object again. (Synchronization object have very little protection for calling them incorrectly, so don't just add an extra release to be sure.)

One word of caution – as with critical sections in general, don't let your critical procedures get too large. For instance, you really wouldn't want to make something like a browse procedure a critical procedure! Stick to short functions that return quickly without user input.

Summary

Shared resources in Clarion 6 aren't necessarily a scary proposition. If you have read-only global data, you may not need to make any changes at all, or the answer may be as simple as using the `THREAD` attribute. For other situations, or if you want to exert control over which threads run when, you'll need to investigate the synchronization objects supplied with Clarion 6. Fortunately, these objects really aren't that difficult to use; most of the battle is figuring out which is the right object for any given situation.

The simplest option is a critical section, which lets you synchronize access to a shared resource within an application (i.e. a process). A critical procedure is a convenient way to apply a critical section to an entire procedure (and with a little work by SoftVelocity it can become a "procedure safe wait" class that will insure a procedure never returns with an object locked). If you need to synchronize access between processes, or you want to use a `TryWait()` function, you need a mutex. If you are managing a number of threads which may or may not need to run at the same time, think semaphore. And if you're looking for a ready-made solution for handling global data, consider the `ReaderWriterLock` class.

Download the source

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

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