Clarion magazine

Main Page
Log In
Subscribe
Open Source
Links
Mailing Lists
Advertising
Submissions
Contact Us
Site Index
ClarionMag FAQ
Download PDFs
Search ClarionMag

Volume 1, Number 8 - September 1999

Issue Index

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

If you're interested take our poll & let us know!

Java does come up for discussion occasionally among Clarion developers, but as a technology it's often misunderstood and underrated. Here's an overview of what Java is and isn't.
Posted on September 21, 1999

Product Review: IFT Server
The Internet Framework Templates make it possible to write web servers and application servers in Clarion. In this first of two related reviews Tom looks at the server template set.
Posted on September 21, 1999

September News
Clarion news and product announcements from around the globe.
Posted on September 21, 1999

DevCon Weekend Edition
DevCon isn't officially under way, but there's already lots happening. Your intrepid reporter brings back text, pictures, and MPEG video!
Posted on September 26, 1999

DevCon '99 Special Subscription Offer
To celebrate DevCon '99 we're offering a special deal to new subscribers: 20 months of Clarion Magazine for $99, a regular value of $125. For $99 you get all the back issues, all the issues with DevCon '99 coverage, and a year's subscription beginning November 1.
Posted on September 22, 1999

DevCon '99 Monday Overview
DevCon '99 Monday Overview Day 1 at DevCon '99 and it's web enable, web enable, web enable. More to come!
Posted on September 28, 1999

TopSpeed's New Direction: The Web
Monday’s focus on web development has been confirmed by Bob Zaunere’s presentation this morning: TopSpeed is making a major directional change toward deploying applications on the web.
Posted on September 28, 1999

TopSpeed Headed For Java
Richard Chapman, TopSpeed's VP of R&D, announced this afternoon that TopSpeed will in a future release generate Java code
Posted on September 29, 1999

Vol 1, No 8
Sept 1999

Clarion magazine

The Cranky Programmer

Install THIS!

I like third party add-ins. No, let me rephrase that – I *love* third party templates and tools.

Being a firm believer in not re-inventing wheels (much less entire engines and transmissions), I thoroughly enjoy acquiring new features for my apps at a fraction of what it would have cost me to do it myself. Assuming that I would even have *had* the time or expertise to do it myself in the first place.

"So," I hear you thinking, "you sound like a pretty happy guy – what's there to be cranky about?"

"Hah!" I reply (with my best Snidely Whiplash sneer).

The worst part of many add-ins is *installing* the dang things. Here is just a partial list of common gaffes and goof-ups that really toast my crackers:

- Not knowing where to install. In other words, deciding that everyone in the world must have Clarion installed on their C drive with the default folder names – so why bother looking?
- Having detailed installation instructions that are installed *during* the installation so that by the time you find them, you are already done. A wonderful bit of circular reasoning, that.
- Not telling you where the add-in put itself. This commonly leads to a search of your menus, all the time wondering -- was it installed under the product name? Under the vendor's name? Is it even on the menu? WHERE THE $(#@&! IS IT!?!
- In a related gripe, vendors assuming that you do want EVERY add-in right up there on your Start menu. This quickly leads to what I call (cue scary music and big echoing voice)... "THE START MENU FROM HELL" – a scrolling monstrosity with a gazillion entries for things that aren't runnable programs to begin with.
- Finishing the install and leaving you wondering what just happened, i.e., not bothering to tell you how to get started with your nifty new toy.

And my all-time favorite:

- Not telling you the name of the template you need to register.

As the old saying goes, there *must* be a better way.

Super-Bruce to the rescue

Bruce Johnson of CapeSoft fame (yes, the one able to leap tall files in a single bound) has put forth a proposal for a standardizing how third-party Clarion tools are installed. This is part of Bruce's larger Clarion 3rd Party Association (C3PA) effort.

I've been testing out some CapeSoft product installations that implement the proposed standards, and I must say that I *like* what I see.

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

etc 2000
If you're interested take our poll & let us know!

Here some of my personal favorites:

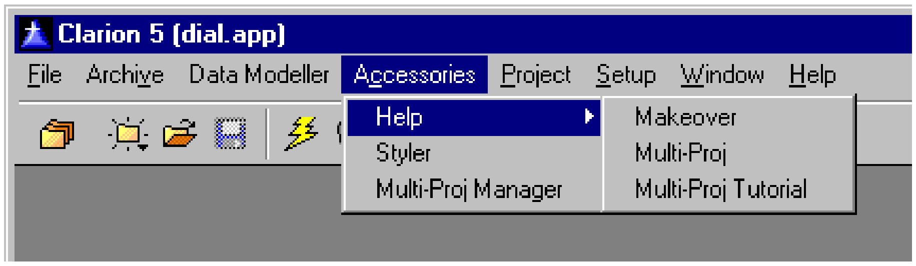
- The root for the install should default to x:\Clarion?\3rdParty, where 'x' is the drive where the appropriate version of Clarion is installed, and 'Clarion?' is the appropriate version of Clarion (CW20, Clarion4 or Clarion5). This means that all of your third party tools will now be in a common location, and that the list of subfolders in your \Clarion folder won't grow so unwieldy. (Cool!)
- All documentation, utilities and a required uninstall for the tool should be placed within a Start menu item called 'Clarion 5 Accessories' (or 4 or 2, etc.). Yeah, baby – no more 'Start Menu From Hell' syndrome. They would still have their own folders, etc., just not at the top Start menu level and they would all be in the same place.
- Note that an uninstall is required – something many tools don't bother with at the moment. It also specifies that the uninstall log should be named something unique to the product. In case you weren't aware of it, some third party products blithely overwrite the Clarion uninstall logs by placing their own log in the Clarion root folder, using the default name of "Install.log." Ouch!
- Products with templates will give you the choice of having them automatically registered. Any other required modifications, such as editing the RED (Clarion redirection) file can also be automated. (Way cool!)
- Utilities with an EXE, etc., would place them in a new Accessories item in the Clarion IDE (you *did* know that you can modify the menus in Clarion, didn't you?). What a concept – putting the tools you need right there at your fingertips where you need to use them!

There are more items on the list, but these should give you a general idea of what Bruce Johnson is striving for: consistent, intelligent, no-brainer installs that put the tools where they be easily used.

Here is an example from within C5, after installing CapeSoft's MakeOver and Multi-Proj products:

Figure 1: My new Clarion5 Accessories menu

Figure 1. My new Clarion5 Accessories menu.



As that famous philosopher Eric Cartman would say... "SWEEEEET!"

I highly recommend that we all join the push to get other vendors involved with this effort – anything that can be done to make installing (and finding, and using) a tool a bit less like a game of Russian roulette is tops in my book.

Makes me just a *wee* bit less cranky, too!

If you would like more information about what Bruce is trying to do, you can visit the preliminary C3PA web site. Note: This is a new effort and Bruce is still putting the site together. If you have suggestions to further enhance the standards, I'm sure he'd like to hear them, too.

So am I happy now? "Hah!" I reply.

See you next time.

Comments? Send them to cranky@clarionmag.com.

Open Source Update
Posted on September 7, 1999

The Clarion Magazine Technology Poll
Posted on September 7, 1999

Correction: Class And Wrapper For Handling Control Files
Posted on September 7, 1999

Vol 1, No 8
Sept 1999

Feature Article

Calling OLE Methods: Part 2

by Jim Kane

Click here to read Part 1 in this series.

If given a task of creating or modifying a file created by another application or the operating system, most programmers would search for the file format and the true meaning of each and every byte in the file. Or at least those who are bit twiddlers from way back and workaholics who never see the sun would do this. If they are lucky enough to find the information, then the task is reduced to a series of API calls to open, read, modify and write the file. Heck, no programmer wants the weekend off anyway!

If the tried and true programmer is successful in reading and writing the file, he or she then either tries to get a new job before the program version changes or gets the rare honor of going back and redoing the code for the new version. At least that's what seems to happen to me.

On the other hand, if instead of getting down into the bits and bytes I could only find some existing code that would take my information in a version independent manner and write or modify the file for me, why I could go home, get reacquainted with my wife and maybe have some time left over for some two-stepping (the national dance of the proud country of Texas).

Well, I guess you could say I'm at the fork in the road. I need to write a Windows shortcut (.lnk) file, and I'm not familiar with the format. There are a few web sites that cracked the file format for .lnk files, but the problem is that this format may change in the future. There is, however, also an OLE interface called `IShellLink` that understands the format and can write the file for me plus relieves me of the burden of version updates.

Fortunately, being a Clarion Magazine charter subscriber, I know there was an interesting article recently that described how to call an OLE interface. While I didn't really understand all that pointer to a pointer to a something or other stuff, it came with the code I need so why don't I just plug it in and see what else I have to do to get the job done once via OLE and let version upgrades take care of themselves.

Most COM programs follow a similar pattern. If I was really clever, I'd probably write a class that would encapsulate the OLE code I use in program after program, but I'm in a hurry so I'll just write it as straight code to get familiar with this new-fangled technology. Maybe next time I'll write it in a reusable fashion provided this COM stuff actually works and I don't need to figure out the magical byte pattern and do it the old way.

After some reading I find that I need to take the following steps to create a Windows shortcut:

1. Initialize COM one time per application (`CoInitialize`)
2. Get my interface pointers (`CoCreateInstance` and

QuerryInterface)

3. Use my interface pointers to create the shortcut (IShellLink Methods)
4. Save the shortcut to disk (IPersistFile Methods)
5. Release any interface pointers I got. (Release method of respective interfaces)
6. Uninitialize COM one time per application (UnCoInitialize)

Since steps 1 and 6 are application global - COM gets initialized once no matter how many procedures call an interface or use COM - I'll put that part in a separate module. I'll call those source files JJKOLE.INC and JJKOLE.CLW to avoid any potential name conflicts. The .inc file will contain prototypes and equates and the .clw file will contain code. Listing 1 shows the code for steps 1 and 6:

Listing 1. Code to initialize / uninitialize COM.

```
  Module('API')
    CoInitialize(long AlwaysNull=0),Pascal,Long,proc
    CoUnInitialize(),pascal
    CoCreateInstance(long AddrClsid, long ClsContext,↵
      long ServerType,long addrIID, *long lpVtable)↵
      ,pascal,long,proc
    MultiByteToWideChar(ulong codepage, ulong dwFlags,↵
      *Cstring MBS,Long LenMBS, long addrWStr, long lenWideStr)↵
      ,pascal,long,proc,raw
  End
  Module('JJKOLE.CLW')
    InitOle(),byte
    KillOle()
  End
  Module('ICall.a')
    ICall0P(long lpVTable, long VtableOffset),long,pascal,↵
      proc,Name('ICall')
    ICall5P(long lpVTable, long VtableOffset, Long p1,long p2,↵
      long p3,long p4,long p5),long,pascal,proc,Name('ICall')
  end

!Data and Code:
fInitComm       byte(0)
!handy dandy equates for return code
Return:Benign   equate(0)
Return:fatal    equate(3)
Return:Notify   equate(5)

InitOle Procedure()
hr Long,auto
  code
  If fInitComm then
     return(Return:Benign)
   end
   hr = CoInitialize(0)
   If hr < 0 then
     fInitComm = False
     Return(Return:Fatal)
   else
     fInitComm = True
     Return(Return:Benign)
   end
```

```
KillOle Procedure()
   Code
   If fInitComm then
     CoUnInitialize()
   end
   Return
```

As you can see there is just one variable involved that prevents the initialization code from being called more than once *and* keeps the uninitialization code from being called if the init wasn't done. With this structure, any COM procedure can call `InitOLE` automatically and if `InitOle` has already been called by another procedure, it just returns success. Success is indicated by a return value of 0. Well that's good news! You've survived the first battle with OLE. You could pop (bring back fond memories of last month's lesson on assembler?) the .inc into the Before Global Includes embed , the prototypes into the global map and add the source module plus Icall.a from last time to the external source section of the project tree and compile. Use 32 bit only please.

To create the shortcut you need to use the `IShellLink` object. When you want to call a COM object, you need to know its name, which is called a `CLSID` (Class ID). These names come from .h (header) files or COM viewer programs. Why `IShellLink` would not do as a name I'll never know, but all COM objects have a unique 128 bit name that can be expressed in Clarion with this specific structure:

```
!IShellLink
ClsId_ShellLink Group
data1             ulong(21401H)
data2             ushort(0)
data3             ushort(0)
data41            byte(0C0H)
data42            byte(0)
data43            byte(0)
data44            byte(0)
data45            byte(0)
data46            byte(0)
data47            byte(0)
data48            byte(46H)
                end
```

(Next time you complain about the name your parents gave you consider the alternative possibilities if Bill had been your daddy!).

The `IShellLink` object contains two interfaces called `IShellLink` and `IPersistFile`. Every interface has a name in the same 128 bit format called an `IID` or Interface ID. As hopefully you recall from our last exciting adventure in assembler land, the other piece of information you need is the offset down the `Vtable` for each method. This can be gotten from a .h file or from .idl, or .old files, or from COM Viewer utility programs (like OLEView which is free from Microsoft's web site) just by counting by 4 in hex down the list of methods. For example, if you look in SHLOBJ.H at the `IShellLink` definition, and copy down the names of the methods in the order they are defined and then count by 4 you get Listing 2.

Listing 2. The IShellLink methods.

```
IShellLink_QuerryInterface      Equate(0)
IShellLink_AddRef               Equate(4)
IShellLink_Release              Equate(8)
IShellLink_GetPath              Equate(0CH)
IShellLink_GetIDList            Equate(10H)
IShellLink_SetIDList            Equate(14H)
IShellLink_GetDescrip           Equate(18H)
IShellLink_SetDescrip           Equate(1CH)
IShellLink_GetWorkDir           Equate(20H)
IShellLink_SetWorkDir           Equate(24H)
IShellLink_GetArguments         Equate(28H)
IShellLink_SetArguments         Equate(2CH)
IShellLink_GetHotKey            Equate(30H)
IShellLink_SetHotKey            Equate(34H)
IShellLink_GetShowCmd           Equate(38H)
IShellLink_SetShowCmd           Equate(3CH)
IShellLink_GetIconLoc           Equate(40H)
IShellLink_SetIconLoc           Equate(44H)
IShellLink_GetRelPath           Equate(48H)
IShellLink_Resolve              Equate(4CH)
IShellLink_SetPath              Equate(50H)
```

This is all a bit tedious, but since you only need to count by four and most people have five digits on an appendage it usually works okay. In case you have not had enough, all the data required for IShellLink is contained in the file IShellLK.inc and all the non-IShellLink specific constants are in JJKOLE.INC. Please have them memorized by noon tomorrow.

Okay, enough tedium! Time for some magic. I'll drop the IShellLink CSLID and IID into the API call CoCreateInstance and produce before your eyes a ppVtable_IShellLink:

```
ppVtable_IShellLink  LONG(0)
!Get the IShellLink ppVtable
   hr = CoCreateInstance( Address(CLSID_ShellLink), NULL, |
        CLSCTX_INPROC_SERVER, Address(IID_IShellLink),|
        ppVtable_IShellLink)
   if hr < 0 then…
```

Since the CLSID and IID are 128 bits or 16 bytes and the stack is only 32 bits wide, these numbers are passed by address. Hence the use of the Clarion Address() function.

Almost all COM APIs and interfaces return a result code often referred to as a Hresult. Here I named it Hr. A value of less than zero is a bad thing. Zero or greater is a good thing. If the result code returned is zero or greater, then the long variable contains the keys to the kingdom of COM: the exalted pointer to a pointer to a vtable, affectionately known here as ppVtable_IShellLink. Most importantly the Icall functions know how to use it.

For my next trick, I'll use the ppVtable to call the QuerryInterface method and get a ppVtable for IPersistFile. This method requires two parameters: the IID of the interface I want, and the address of the variable to store the ppVtable in should the function call succeed. Seems fair enough, so let's do it.

From IShellLk.clw...

```
hr = ICall2p(ppVtable_IShellLink, IShellLink_QuerryInterface, |
  Address(IID_IPersistFile), Address(ppVtable_IPersistFile))
   If hr < 0 then….
```

I use my Icall function from last time with the ppVtable and offset in my .inc file. Again the hresult is returned and I test for errors. Boy is this getting old fast! Wouldn't it be nice if I could write the code to do the error checking and orderly close down once and use it many times? Maybe DAB knows a way to write once and reuse. I'll ask him before next time.

Now that all the interface pointers have been obtained they can be used to call the "set" methods of IShellLink. The data passed to my COM procedure is pretty mundane (vocabulary word from my Garfield calendar) except for one call SpecialLocation. Say you want your shortcut to appear on your user's desktop. The directory structures in Windows '95 and higher for different users and different desktops get a little wild. Fortunately COM can refer to special places like the desktop, recycle bin, and My Computer with other than a directory name. That way, no matter how the user's computer is configured or what the drive letter, the created shortcut gets to the correct place. Once again you would think the simple string 'DESKTOP' would be an adequate identifier but brother Bill says "unless I make it complicated every one would be a programmer." So he set up a list of equates called CSIDLs (not CLSIDs - try keeping that straight) for all the special places. The equate for CSIDL_DESKTOP happens to be 0. If a SpecialLocation is passed you use the pair of APIs:

```
!PIDL = pointer to a Identifier List
hr = ShGetSpecialFolderLocation(0,SCS.SpecialLocation,lppidl)
hr = ShGetPathFromIDList(lpPidl,szPath)
```

to convert the SpecialLocation code to first a PIDL and then onto a CSTRING path called szPath. Each step of the way you do some error checking on the hrs returned. Although perhaps one of the goofiest names in all of COM, PIDLs are fairly important in the windows shell API. The Win95 and higher shells, and Explorer in particular, show everything under the sun as if it was a file in its tree structure. That includes "special places," printers, disk drives...you name it.

While files have a path and name to identify them other objects may not. PIDLs were created to provide an identifier for everything Explorer or the shell has to deal with. While it may be a bit of overkill for files, it does serve a purpose as a generic identifier. The functions listed above provide conversions from a special location equate to PIDL and from PIDL to filename. In Clarion, a pointer to a IDL or a PIDL can simply be prototyped as a LONG.

In any case, whether a CSTRING with the .lnk file name was passed or whether a special location equate was passed, I now have the path to store the .lnk at the location contained in the variable szpath. In this case that location is the desktop.

Unfortunately COM uses some data types that Clarion programmers may not yet be familiar with. Common COM data types include BStrings, Wide Strings, and Variants. I'll save a discussion of these data types for another day but the more you use COM the more you'll run into these data types. It sure would be nice to have some reusable code to convert from Clarion data types to COM data types and back....

For today though I'll just take the CSTRING for the path szPath and convert it to a Wide String called wszLinkFile. The CP_ACP means use the ANSI code page. I have no idea what MB_Precomposed means but it always works.

!Now convert to a wide string:

```
MultiByteToWideChar(CP_ACP, MB_PRECOMPOSED, szPath, -1, |
  Address(wszLinkFile), File:MaxFilePath);
```

Lastly I take my wide string, ppVtable for IPersistFile, and the offset to the SAVE method and save the lnk file who's path is stored in wszLinkFIle to disk where it belongs. Then I test the hr code. I can hear that two-step music starting up now! Almost done.

```
!Save the shortcut to disk then cleanup and exit:
hr = ICall2p(ppVtable_IPersistFile, IPersistFile_Save, |
 Address(wszLinkFile), True)
! The True above means clear the file is dirty flag.
! It would be false if this was a SaveAs
if hr < 0 then...
```

If at any time during the procedure I get an hr code of less than zero *or* at the end when I'm done (Step 5) I need to call the Release method of any interface for which I got a ppVtable. COM uses a simple counting method to know when all users are done with an object. Addref is automatically called for you when you call CoCreateInstance or QueryInterface to get the exalted ppVtable, but when done you should be a nice person and call Release so the COM code, usually in a DLL, can be unloaded as soon as possible to conserve resources. In any case, calling Release is easy. No parameters are required (which kind of explains the use of CallOP); just call with the ppVtable you are done with and the Vtable offset for Release. The Vtable offset for Release is always 8 since it is always the 3rd method. (All COM interfaces start with QuerryInterface (+0), AddRef(+4) and Release(+8).)

```
if ppVtable_IPersistFile then
  iCall0P(ppVtable_IPersistFile,IPersistFile_Release)
end
If ppVtable_IShellLink then
  iCall0P(ppVtable_IShellLink,IShellLink_Release)
end
```

If I had stored my ppVtable in a Queue I could really automate this cleanup stuff since xxx_Release is always 8H...food for thought.

Well, there you have it. Load up the ShortCutStruct defined in IShellLk.inc with the address of the string data and other data you want for your .lnk file and call the CreateShortCut function I just wrote. The reusable once per app code is in jjkole.inc/clw and the IShellLink specific code is in Ishelllk.inc/clw. The code to glue the demo all together is in Shortcut.clw. I decided to use .prj file rather than an .app file so I would not be Clarion version specific. Listing 3 shows the demo code and instructions for putting it together. If you prefer to use AppGen, the equivalent embed points are presented.

Listing 3. The demo code.

```
Program
  Map
   ! Add the needed prototypes in the global map
   Include('JJKOLE.INC','OLEPROTOTYPES') !General OLE Prototypes
   Include('IShellLk.inc','IShellLkProto') ! IShellLink prototypes
  end

! Add the needed equates
! In the After Global Includes embed
Include('JJKOLE.INC',  'OLEEquates')
include('IShellLk.inc','IShellLkEquates')

! JJKOLE.Clw, and ICall.A have been added to the project
! These are required in all OLE apps
! IShellLk.Clw is also added - this has IShellLink Specific code
! for creating shortcuts

! Data for the test
ShortCutStruct  like(ShortCutStructType)
Target          cstring('C:\Windows\Notepad.exe')
Desc            cstring('Description')
LinkFileName    cstring('NewNotePadLink.LNK')
WorkingDir      cstring('C:\Windows\Temp')

  code

  ! Load up the structure with the data declared above and you're off
  Clear(ShortCutStruct)

  ! Required Parameter pgm to run when shortcut clicked
  ShortCutStruct.lpTarget           = Address(Target)

  ! Optional descriptions - this text is NEVER
  ! seen by the user - It's a secret.
  ShortCutStruct.lpDesc             = Address(Desc)

  ! Required Parameter
  ShortCutStruct.lpLinkFileName     = Address(LinkFileName)

  !Optional working directory
  ShortCutStruct.lpWorkingDir       = Address(WorkingDir)

  ! Required, Take icon from target file, iconindex=0
  ShortCutStruct.lpIconPath         = ShortCutStruct.lpTarget

  ! Let's put it on the desktop
  ShortCutStruct.SpecialLocation    = CSIDL_DESKTOP
  If CreateShortCut(ShortCutStruct)
     Message('Create Shortcut Failed')
  else
     Message('Create Shortcut Worked')
  end

  ! Any number of other OLE actions - more shortcuts or what ever

  KillOle()
```

```
Return
```

Summary

Rather than slugging out the file structure of a .lnk file and writing the .lnk using the brute force method, I invoked the operating system interface for creating a shortcut and let the operating system handle the details of the file.

The rules of COM (or OLE: the two are synonyms - thought I'd wait to the end to tell you that) are fairly simple. Initialize COM at the start and uninitialize when done. Get all the interface pointers needed but for every one you collect when done call its `Release` method at offset 8. Once you have interface pointers, use the `Icall` code and offset to the particular method you want to in order to call the interface method of your dreams. Before a call to an interface method, convert from Clarion data types to COM data types. After a call to an interface method either ignore the `hresult` code if its not important or test to see if its less than zero indicating an error. On error or when done, clean up thy mess.

If you've gotten this far and are still awake, you've come far. You've taken the seemingly insurmountable goal of calling a COM interface and you've worked through the methodology. All you need now is the raw ammo of `CLSIDs`, `IIDs`, `CSLIDs`, and Vtable offsets and some reusable code for error checking, cleanup, initializing com, getting `ppVtables`, and converting COM to Clarion datatypes back and forth. I'll save that for another day. Time to go dance!

Download the source code

Jim Kane was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and since graduating has served in the US Air Force. He is currently the Officer in Charge of Dental Facility Design at USAF Dental Investigation Service in San Antonio, Texas. In his spare time, he runs a computer consulting service, Productive Software Solutions, which he hopes to run full time after retiring from the US Air Force Dental Corps in June 2000. He is married to the former Jane Callahan of Cando, North Dakota.Jim and Jane have two children, Thomas and Amy.

Vol 1, No 7
August 1999

A Class And Wrapper For Handling Control Files

by Nik Johnson

I'm an inveterate template tinkerer. It's a mixed blessing. By the time I abandoned my CPD 2.1 model file for the Logix Project Manager, there was more logic devoted to branching conditionally around various options than to perform the actual task at hand. By the time CDD 3007 rolled around, I had so much invested in modifying Todd Carlson's templates that I didn't dare switch to newer Clarion templates. But while they lasted these modified tools made me a lot more productive than I would otherwise have been.

I'm still tinkering, but the introduction of OOP and the ABC library has meant that I can now work *within* the TopSpeed framework rather than *around* it. Steve Parker's article on control files in the previous issue provides an opportune way to illustrate the convenience and power of OOP/ABC while at the same time extending my toolkit.

Defining The Problem

What I want to build is a "set and forget" method of handling control files. As Steve points out, some file structures require a `SET/NEXT` approach, others do better with `GET/POINTER`. Still others may require other access methods. I want to spend an absolute minimum of time and effort incorporating control files into future applications, independent of the file system.

Steve has enumerated the things our tools need to be able to do:

- Open the file
- If the file is missing, optionally create it
- Read the file's only record in a way appropriate to the file structure
- If the file has no records, add one
- Update the record
- Close the file

The following narrative mimics the way I go about building tools for my own use. First, list the things that have to be done; second, establish a general design; third, build a skeleton; fourth, fill in the details. This works for a small shop, but if you're building tools for sale or working in a larger organization you may want to skew this pattern toward heavier documentation of the design in advance of coding.

This Wheel Has Already Been Invented ... Almost

The ABC FileManager class provides facilities to accomplish five of these six tasks, so it makes sense to start with that as a basis. A class based on `FileManager` can inherit all of its functionality and minimize the new work that needs to be done.

```
MyFileManager CLASS(FileManager)
Fetch           PROCEDURE,BYTE,PROC
              END
```

The new method provides a place to add Steve's access logic, but it doesn't include provision for specifying which version of that logic to use. ABC classes are insensitive to driver, but in this case that luxury is not available.

Two possibilities come to mind. First, the fetch algorithm could accept a flag to designate how access to the one and only record in the file is to be gained. A byte should be sufficient, since more than 255 variations on the get-only-record theme are unlikely. However, if the file driver changes, it could be inconvenient to chase down every `Fetch` and alter it. The second approach, adding a property to the derived file manager, permits the fetch algorithm to be specified once at the beginning of the program.

```
MyFileManager CLASS(FileManager)
FetchOnlyType   BYTE
Fetch           PROCEDURE,BYTE,PROC
              END
```

The compiler can distinguish between the new `Fetch` method and the one specified in the standard ABC FileManager class because their prototypes can be differentiated by the rules for procedure overloading. (See pp. 89-90 in the Language Reference Manual.)

Building A Skeleton

A residence for record fetching logic having been established, the logic itself can be added.

First, set up an `EQUATE` for each value `MyFileManger.FetchOnlyType` can assume:

```
            ITEMIZE,PRE(FetchMethod)
GetByPosition EQUATE
GetNext       EQUATE
            END
```

Although these are the only two options at the moment, setting up this structure provides a clean way to add other options in the future.

The new `Fetch` method, by using the `EQUATE`s, becomes self-documenting.

```
MyFileManager.Fetch PROCEDURE
ReturnValue BYTE,AUTO
  CODE
  ASSERT(SELF.FetchOnlyType)
  SELF.Open
  SELF.UseFile
  CASE SELF.FetchOnlyType
  OF FetchMethod:GetByPosition
    ! "get by position" code here ...
  OF FetchMethod:GetNext
    ! "get by set/next" code here ...
  END
  IF ReturnValue
```

```
    CLEAR(SELF.File)
    ReturnValue = SELF.Insert()
  END
  SELF.Close
  RETURN ReturnValue
```

This "shell" contains everything except the actual logic to access the configuration record if it exists. In adapting the logic from Steve's article, I've made a few assumptions based on my own expected use of the method:

- I may want to access the configuration record in a context other than an update form, so I've removed logic that refers to "SetupForm" from Steve's code.
- I should never call this method unless I've set the FetchOnlyType property. The ASSERT protects against this. Since this would be a programming error as opposed to a data-related condition, I don't need user-friendly error messages for this situation.
- If I ask for a record and can't find one, I always want to add the record.

Your programming style may suggest a different set of design assumptions.

Putting Meat On The Bones

The only thing left to do is add appropriate code for accessing the configuration record. Whatever the method used, `ReturnValue` should be set to zero if the fetch is successful, something else if not. The `CASE` structure is the only part of the shell which changes:

```
CASE SELF.FetchOnlyType
  OF FetchMethod:GetByPosition
    GET(SELF.File,1)
    IF ERRORCODE()
      ReturnValue = Level:Notify
    ELSE
      ReturnValue = Level:Benign
    END
  OF FetchMethod:GetNext
    SET(SELF.File)
    ReturnValue = SELF.Next()
END
```

The new method behaves very much like the standard ABC `Fetch` except that it doesn't require specification of a key, it expects one and only one record in the file, and, if the file is empty, it adds a cleared record.

Adding The New Class To The ABC Library

A little plagiarism is a wonderful thing. All the information needed to make making the new class look to Clarion like an ABC class is sitting in the \LIBSRC directory. First, set up files for the new class prototypes and methods. Call them something like `MyClasses.inc` and `MyClasses.clw`. Referring to similar files shipped with Clarion, set up these two files to match their style.

Here's the general setup of MyClasses.inc:

```
!ABCIncludeFile
 OMIT('__EndOfInclude__',_MyClassesPresent_)
_MyClassesPresent_ EQUATE(1)
    INCLUDE('ABFILE.INC'),ONCE
 ! class prototypes here …
__EndOfInclude__
```

The comment (!ABCIncludeFile) tells the IDE that this code follows the ABC pattern and should be treated as any other ABC include file. The OMIT structure lets this file be included anywhere the definitions are needed without fear of duplicating those definitions. For example, the definitions of the ABC FileManager are included above so that they can be used in the definition of the MyFileManager class.

The new ONCE attribute provides another mechanism for avoiding duplicate definitions, but that protection depends on the ONCE attribute appearing in every INCLUDE. Older code may still require the protection provided by the OMIT structure, so it's a good idea to leave it in place.

A little more creative plagiarism provides the general setup of MyClasses.clw:

```
MEMBER
_ABCDllMode_  EQUATE(0)
_ABCLinkMode_ EQUATE(1)
  MAP
  END
  INCLUDE('MyClasses.inc')
```

The MEMBER statement identifies this as a source module, something the compiler needs to know. The MAP structure is required, since it causes the compiler to include prototypes for Clarion language statements and functions. The INCLUDE of MyClasses.inc makes the new class definitions available to code in this module and, if the nest is not too deep (LRM page 95), also includes definitions from ABFILE.INC.

The two equates, for _ABCDllMode_ and _ABCLinkMode_, implement the ABC library's method of determining where and how these methods will be linked and referenced. These definitions work with attributes in each CLASS statement which are required and will be added next.

In the skeleton version of the CLASS prototype, some necessary attributes were ignored. The full statement should have been:

```
MyFileManager CLASS(FileManager),|
                    TYPE ,  |
                    MODULE(‘MyClasses.clw'), |
                    LINK(‘MyClasses.clw',_ABCLinkMode_), |
                    DLL(_ABCDllMode)
```

The TYPE attribute identifies this class as a prototype only. Actual instances of the class will be created (instantiated) when needed. The MODULE attribute tells the compiler where to find the code for the class methods. The LINK attribute tells the compiler to compile and link these methods if, and only if, _ABCLinkMode_ is True. The DLL attribute tells the compiler to look for these methods in another DLL if, and only if, _ABCDllMode_ is True.

The full code for MyClasses.inc and MyClasses.clw is available for download at the end of this article.

Have Hammer, Need Nail

Having built a new class, the next challenge is to use it. There are at least two likely situations:

- Access within the context of some other process
- Access in conjunction with an update form

The first type of use is easy. Just use the Fetch method as you would the standard ABC Fetch:

```
IF NOT Access:MyConfigFile.Fetch
  ! do something, possibly including ...
  Access:MyConfigFile.Update
END
```

(This code assumes that nothing goes wrong during the update. You are of course free to be as conservative as the situation warrants.)

The second is even easier. In the ThisWindow.Init method of a form, before the code stores GlobalRequest, insert:

```
IF Access:MyConfigFile.Fetch
  RETURN Level:Fatal
ELSE
  GlobalRequest = UpdateRecord
END
```

This allows the form to be called directly from a menu and, no matter how it is called, causes it to update the one and only record in our configuration file.

To make this capability available for a particular file, set Access:MyConfigFile.FetchOnlyType to a value indicating how the single record should be accessed. This can be done anytime after the FileManager instances are initialized and before the Fetch is called.

Building A Wrapper

It would be nice if all of the necessary housekeeping could be wrapped up in a simple package so that implementing a configuration file would require nothing more than, say, adding a word to the user options of that file in the dictionary. Making things that simple is probably overkill, though, since most projects will have only one file of this type.

An application level extension template can do the same thing cleanly and without the extra processing needed to check every file in the dictionary for a user option. Given the name of a file, the template can initialize the corresponding file manager's FetchOnlyType property and add setup code to any form procedure which uses the file.

This is as good a time as any to set up a file for home grown templates. Call it MyTemps.tpl. A single line identifies the template chain:

```
#TEMPLATE(MyTemps,'Homegrown Templates'),FAMILY('ABC')
```

Another line establishes the extension template:

```
#EXTENSION(ConfigFile,'Implement Configuration File’),APPLICATION,MULTI
```

The APPLICATION attribute tells the generator that this extension is applied at the application level rather than the procedure level. The MULTI attribute permits more than one instance of the extension in a single application.

A couple of lines of documentation describing what this template is supposed to do are in order:

```
#DISPLAY(‘This extension adds code to handle a specified file’)
#DISPLAY(‘as a configuration file containing one and only one’)
#DISPLAY(‘record. Forms for which this file is the primary file’)
#DISPLAY(‘can be called directly without an intervening browse.’)
#DISPLAY(' ')
```

A prompt allows specification of the file:

```
#PROMPT('Configuration File:',FILE),%MyConfigFile,REQ
```

There are at least two ways to specify the access method. The easiest is to add a second prompt (or more accurately, prompt structure):

```
#PROMPT(‘Choose Access Method:’,OPTION),%MyAccessMethod
#PROMPT(‘Get by position’,RADIO)
#PROMPT(‘Get next’,RADIO)
```

Another alternative is to have the template select the access method based on driver. This has the advantage of hiding the access method from the programmer, thereby reducing the number of things that have to be remembered when using the template. A good place to put this selection logic is in an #ATSTART section:

```
#ATSTART
  #FIX(%File,%MyConfigFile)
  #CASE(UPPER(%FileDriver))
  #OF(‘CLARION’)
  #OROF(‘DBASE3’)
  #OROF(‘DBASE4’)
    #SET(%MyAccessMethod,’Get by position’)
  #OF(‘TOPSPEED’)
    #SET(%MyAccessMethod,’Get next’)
  #ELSE
    #ERROR(‘File driver not recognized by ConfigFile extension’)
  #ENDCASE
#ENDAT
```

At this point everything necessary to generate code is in place. First, provide for initialization of Access:MyConfigFile.FetchOnlyType:

```
#AT(%ProgramSetup)
  #CASE(%MyAccessMethod)
  #OF('Get by position')
Access:%MyConfigFile.FetchOnlyType = FetchMethod:GetByPosition
  #OF('Get next')
Access:%MyConfigFile.FetchOnlyType = FetchMethod:GetNext
  #ENDCASE
#ENDAT
```

Adding code to the input form is a little trickier. The basic structure is easy:

```
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE'),PRIORITY(0)
IF Access:MyConfigFile.Fetch
  RETURN Level:Fatal
ELSE
  GlobalRequest = UpdateRecord
END
#ENDAT
```

Specifying PRIORITY(0) puts the code at the very beginning of the method, which is necessary because GlobalRequest is internalized very early in the initialization process.

This code should only be generated if the window is an update form for MyConfigFile. The template language provides a WHERE attribute to allow this kind of selection. With that attribute in place, the #AT statement becomes:

```
#AT(%WindowManagerMethodCodeSection,'Init','(),BYTE'),    %|
                     PRIORITY(0),                         %|
                     WHERE(%ProcedureCategory = 'Form'    %|
                            AND %Primary = %MyConfigFile)
```

One of the benefits of writing articles is that you learn things in the process. Until I had to split a template instruction to fit the printed page I was unaware that the template language has a line continuation symbol and that it differs slightly from the continuation symbol used in Clarion code. You'll find it documented on page 503 of the Programmer's Guide.

Unfortunately, when I tried to use this syntax in an example, the registry rejected the template. You'll find a note which suggests that this might happen on page 525 of the Programmer's Guide. So the continuation symbols (%|) in the above example are there for readablity only and *should not be used in actual code*.

The point here is not that the documentation is bad. In fact, it's very good. But the template language has always had more little vagaries and nuances than the Clarion language itself, which makes documentation a daunting task. With both template and Clarion code, I read the documentation, code accordingly, test, and modify until it works. But with templates I don't worry too much if I can't explain in detail why what works, works.

Testing also reveals that %Primary, a built-in symbol which the documentation suggests should contain the label of the procedure's primary file, is blank when the WHERE clause is evaluated. Why? I don't know. But replacing %Primary with %File, a multi-valued built-in symbol, works. Why? I don't know.

Applying The Tool

The three files which define the MyFileManager class and the ConfigFile extension are included in a downloadable ZIP file. Using them is very simple:

- Place MyClasses.inc and MycClasses.clw in your Clarion LIBSRC subdirectory.
- Place MyTemps.tpl in your Clarion TEMPLATE subdirectory
- Register the ConfigFile extension.
- For any file that you want to handle as a configuration file, add an instance of the extension to your application's GLOBAL properties.

You will also need to tell the generator to use MyFileManager instead of FileManager for any configuration files. You can do this either on the Individual File Overrides tab or the Classes tab. The difference will be whether all files use MyFileManager (Classes tab) or just the ones you want handled as configuration files (Individual File Overrides tab).

I usually choose the more general case, expecting to add other functionality to the derived file manager.

In summary, once you have found a solution to a particular coding problem, it makes sense to take the extra step and build tools which implement that solution. Clarion's tool-building facilities are within the abilities of the average programmer, and skill in using those facilities improves rapidly with practice. I hope this example not only proves useful to you, but tempts you to build other tools on your own. You have nothing to lose and a world of productivity to gain.

A Correction

After this article first appeared, I received an email from Jan Jacob de Maa. He had downloaded the associated ZIP file and was trying to use it. Unfortunately, he was encountering compile errors.

Jan Jacob's experience led to the discovery of three errors in the article, which errors are repeated in the ZIP file:

- In the header for the `MyFileManager` class, an underscore is missing in the DLL attribute. It should read `DLL(_ABCDllMode_)` rather than `DLL(_ABCDllMode)`. The missing underscore causes the `DLL` attribute to remain off when it should be on, wreaking havoc when compiling and running in 32-bit mode.
- In the wrapper template, the line `IF Access:MyConfigFile.Fetch` is missing the parentheses necessary to tell the compiler that this is a function rather than a variable. It should read `IF Access:MyConfigFile.Fetch()`.
- Also in the wrapper, the priority of zero which was given for code to be inserted in `ThisWindow.Init` causes the wrapper to execute its `Fetch` operation before the file is open. Changing the priority to `8000` places the code correctly. Also, because at this new position `GlobalRequest` has already been internalized, `GlobalRequest = ChangeRecord` has been changed to `SELF.Request = ChangeRecord`.

Download the updated source code

Nik Johnson stumbled into the programming racket in 1959 when his boss at Grumman Aircraft insisted on his attending a Fortran class. Since 1986 he has been using Clarion to help clients solve information handling problems.

Vol 1, No 7
August 1999

CoveComm Inc.

Clarion magazine

Reader Poll

Thank you for your interest in Clarion Magazine's Technologies survey. The poll is now closed, and results will be posted in an upcoming issue.

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

In This Issue

Clarion magazine

The Clarion Open Source Project

Product Index

Updated September 20, 1999

The following products are available from Clarion Magazine under the Developers Open Source Public License (DOSPL).

- Debugging/Profiling - These classes from CoveComm Inc. generate trace logs and application execution logs. Template included. Version 1.1.
  Posted Aug 3, 1999
- BitList Management - Class and template from JS Software (Jeff Slarve) to bitslice LONGs and display checkboxes for setting the individual bits. Version 0.9912. Very slick.
  Posted August 5, 1999
- File Utilities - A set of classes by Patrick O'Brien to handle file dialogs and splitting file names. Includes an example application.
  Posted August 7, 1999
- Debug Classes - Richard Rogers' class and template for debugging/tracing.
  Posted August 9, 1999
- Graphing Classes - Chris Behling's example application does graphing using all-Clarion code.
  Posted August 19, 1999
- Trackbar/Slider - Pierre Tremblay's 32-bit slider class and template as described in his Clarion Magazine article.
  Posted August 19, 1999
- Graphing Learning Example - James Cooke's learning example does OOP graphing using all embed points.
  Posted September 5, 1999
- NEW! Wizatron Style Viewer - a database application for viewing/managing style sheets. 386k, includes EXE
  Posted September 20, 1999

If you've placed some code under the DOSPL and you'd like to have it listed here, email cosp@clarionmag.com

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

etc 2000
If you're interested take our poll & let us know!

In This Issue

Vol 1, No 8
Sept 1999

Feature Article

ABC Design Series: The RelationManager Class

by David Bayliss

In my series of articles on the FileManagerclass I explained that the FileManager was logically there to embellish the underlying file drivers with information from the Clarion dictionary. The RelationManager class takes this dictionary embellishment one stage further to add the notion of related files. Currently there are three features this brings to the table:

- Referential Integrity. It is quite possible for a file to be physically correct, pass the file level validation constraints, and yet still not correctly relate to the other files. The RelationManager therefore duplicates a number of file access functions, and the use of the RelationManager versions of these functions ensures that the file is correctly linked to other related files.
- File Unification. This allows primary files which are linked to secondary children to be treated as a logical unity. This is a concept I occasionally refer to as BILF management (BILF stands for Bloomin' Irritating Little Files). A primary file could contain 100 fields, 10 of which are linked to children. Yet those child files don't actually *mean* anything; they are just created as part of the data normalisation process. It is really ugly if every time you use a BILF you have to go throughout your code opening it, preserving it, etc. The RelationManager therefore replicates some FileManager functions where the only service it performs is to perform the action upon all the related files in the tree.
- Information provision. Other parts of ABC sometimes need to know information about relations (notably linking fields and keys). The RelationManager provides a portable interface to this information.

Considerations

To some extent all the considerations mentioned in the FileManager articles apply to the RelationManager, although less so. The RelationManager is built on top of the FileManager; specifically there is a one-to-one instance link between RelationManagers and FileManagers. As such the RelationManager always tries to use a FileManager function for a given activity if it can. This is not sheer laziness. By utilising the FileManager, any overriding of the FileManager automatically works for code using the RelationManager.

There were a couple of new issues too. One was sheer complexity (and thus the need for safety). The legacy relational integrity (RI) code went through at least a couple of iterations and to this day it still falls over some cases and corrupts file buffers at will. For ABC we wanted an RI system that was rock solid, but also efficient. Legacy had another problem that for large dictionaries (especially heavily related ones) the code bloated horribly, and we wanted to reduce that drastically.

Further we wanted (in the future) to be able to extend the system to allow one-to-one and many-to-many relationships. Finally we wanted the RI code to simply drop away if it is handled by the back end (usually on an SQL database). That's a pretty long shopping list!

As I head through the code overview I will warn you that the RI methods are by far the most complex procedures in the *whole* of ABC. They are an interesting example of my belief that you should isolate complexity. Don't smear it throughout code (where everyone can stumble over it) but focus it into a small space that you can approach with caution. Well, here are six small procedures (the largest is 60 lines) that get the Bayliss classification of *ice pack jobs*. It is my job to make them clear enough that everyone (at least everyone who is prepared to try) can understand them. I hope I succeed. For the sake of brevity I shall assume that you have read the FieldClass design documents.

Coffee... Icepack .... Action .... (On the plus side, if you can handle this then you are over the ABC learning curve. From here it is just more, not harder).

I strongly urge you to have the source code to hand whilst going through this article; it really will make everything a bit clearer.

Initialisation

The file drivers have no knowledge of the relationships provided in the dictionary; for this reason all the relation information has to be provided by the templates to the base classes. This is done by the templates overriding the `.Init` method and making a succession of `Addxxxxx` calls.

```
AddRelation PROCEDURE(RelationManager RM),PROTECTED
```

A Clarion relation can be viewed from either end and it is not enforced that both directions have a key (although you do need a key both ways for RI). This `AddRelation` method is called when the file being initialised is related to the file being passed in but where there is no linking key on the file being passed in. You may prefer to look at this as saying "he is related to me."

```
AddRelation PROCEDURE(RelationManager RM,BYTE UpdateMode,BYTE DeleteMode,↵
      KEY His),PROTECTED
```

This method gives the ability to note a fully fledged relationship. The `RelationManager` passed in denotes the related file, `His` is the key you fill to get at his data. `UpdateMode` and `DeleteMode` specify the action to be taken upon a potential RI violation.

This `AddRelation` method has an interesting side effect: it primes the object to start accepting `AddRelationLink` method calls. There are OOP purists I know well (some I work with) who frown upon this kind of state within an object (the problem for the purists being that AddRelation *must* be called before AddRelationLink), but pragmatically it is efficient and encourages the object user to write readable code. What is actually happening is that this `AddRelation` creates a `BufferedPairsClass` which will then be filled with the linking fields of the relation.

```
AddRelationLink PROCEDURE(*? Left,*? Right),PROTECTED
```

There are two other `AddRelationLink` functions besides this one, but the variations are simply there to save code size. (A *? parameter takes about 50 bytes of code to pass, `*LONG` parameters take four bytes, `*STRING` parameters take six. Given that `LONG` and `STRING` cover 90% of all linking fields this efficiency is worth having.) What is going on here is simple, but needs grasping. This method is called from the templates with something like:

```
Relate:File1.AddRelationLink(File1.KeyField1,File2.KeyField1)
```

The *? parameter means the *address* of these fields is passed in and squirreled away for future use. Once this has been done for all the linking fields it is possible to assign from one set of linking fields to another using a single statement.

```
Init PROCEDURE(FileManager FM,BYTE UseLogout=0)
```

The base `Init` method simply ties in the `FileManager` this `RelationManager` is based upon. It also creates a queue for the relations and sets an internal property to denote whether transactions are to be framed within `LOGOUT/COMMIT` sections. Remember however that in template usage the `Init` method will typically be derived (in generated source) and the derived method will be full of calls to `AddRelation` to describe the dictionary fully within ABC.

If used fully this approach gives tremendous flexibility. It is quite possible to add files into the RI tree/or cut them out dependent upon system configuration. For example, you could have a file that is only shipped to certain customers but which is in an RI chain *if* it is shipped.

```
Kill PROCEDURE,VIRTUAL
```

This method simply steps through the relation queue, killing off any `FieldPairs` classes that have been created (for the `RelationLinks`) and then disposing them.

```
SetAlias PROCEDURE(RelationManager RM)
```

This method is used to specify that the current `RelationManager` is managing an alias of the passed in `RelationManager`. This method doesn't really do anything; it is simply there to enable the `AliasFile` property to be private. I didn't want the property public as I expect it to die when the `FileClass` comes along.

FileManager Replacements

These are substitutes for the `FileManager` equivalents. As such their basic semantics are the same. The difference is the related files are taken into account. For ease of explanation I am not tackling these in alphabetical order.

```
CancelAutoInc PROCEDURE(),BYTE,PROC,VIRTUAL
```

This method enables the form to readily tackle the problem of orphaned child records. (See FileManager III). The form can simply call the `RelationManager` equivalent (you should always consider `Relate:File.Thing as "Access:File.Thing(Taking into account related files)"`). The `RelationManager` calls down into the `FileManager` (passing in itself) to ensure children are taken care of.

```
Close PROCEDURE(BYTE Cascading=0),BYTE,PROC,VIRTUAL
```

This method simply issues a FileManager close on the current file, and all the child files, grandchild files etc. You would think this is quite easy, and in principle it is, but there is one little gotcha that makes the code quite complex. First consider the logical implementation. To Relate-Open file Fred you first open Fred then you open all of Fred's children. Then somehow you need to get the children to open their children.... Hang on, that's easy. Instead of opening Fred's children, you Relate-Open them and it all works. So a simple recursive solution would be:

```
RelationManager.Close PROCEDURE
I BYTE,AUTO
  CODE
  ASSERT(NOT SELF.Relations &= NULL)
  SELF.Me.Close()
  LOOP I = 1 TO RECORDS(SELF.Relations)
    GET(SELF.Relations,I)
    SELF.Relations.File.Close(1)
  END
```

Beautiful, elegant, efficient and liable to lock your machine the first time you try it. Imagine you have relationships A <- >> B < - >> C <->>D and A <->>D. Technically this is illegal in the Clarion paradigm (you need an alias for the second usage of D) but in practise you can usually get away with this (few procedures will have A, B, C and D all populated) and peoples dictionaries are littered with cyclic dependancies.

Now the recursive solution dies horribly. Suppose you close A. This closes B which closes C which closes D which closes A which closes B which closes.... You get the picture.

There are many sophisticated and elegant algorithms for detecting loops in graphs; we opted for a simple one. The idea is roughly this: when you get the first (top-most) call to close then you note the time. You then recurse as before but when you do the close you note inside the `RelationManager` the time you did the close. Then when you call a `RelationManager` to close it, you see if it has been closed since (or at) the top-most call. If it has then you have already been here before so you exit without recursing. You can actually implement this using `CLOCK` but there is one more little trick to spot. You don't have to use real time; any time will do. So for efficiency I made my own time stored in the `Epoc` variable. This time only ticks when the top-most call is made.

Here's a look at the code. First I check the cascading flag. This flag is purely there to indicate whether this is the "top" of the tree. If it is the top of the tree (cascading false) then I increment the `epoc` timer, if not then I check if for a touch in this "time-zone." If there has been a touch then the code returns; if not then I update the "last-touched" to prevent further recursion. Then it is just a case of closing this file, and then stepping through the children closing them. One extra tweak is an *early out* mechanism. Essentially if any of the `FileManager.Close` calls fail the tree walk stops. This is not particularly useful in the `Close` case but in general a `FileManager` method returning an error could easily have put up an error message to the user. If that has happened once the last thing the user wants is to step through error messages for each of the 150 related files as well.

```
Open PROCEDURE(BYTE Cascading=0),BYTE,PROC,VIRTUAL
```

The Open code is actually very similar to Close. I'm surprised I didn't use a parameterised private method—watch this space, as it is possible Open and Close will both have become shells for an OpenCloseServer by the time you read this. As an aside, I wonder if that seems unprofessional to you? Making mistakes, owning up to them and go fixing them? I never cease to be amazed by the people who write their code badly and then consider it inviolable. Encapsulation, a key feature of ABC, enables us to get the code right. Not *OK*, not *working* but *right*.

The one tweak is the LazyOpen mechanism. The FileManager has an attitude that says it won't actually open a file just because you asked it to. However we felt is reasonable that the primary file *should* be opened straight away so if this is the top of the open call tree (and cascade is thus 0) we call UseFile to force the file open.

```
Delete PROCEDURE(BYTE Query=1),BYTE,VIRTUAL,PROC
```

This method is the first of the nasties. Delete is really just there to delete the primary record. There are two main complications: the first is the need to check that you can delete the primary record (i.e. there are no RI constraints), and the second is the need for transaction framing (the ability to abort the delete process halfway through if something goes wrong and you need to undo all the mess you made).

First is a fairly simply query as to whether or not the user actually wants this record deleted. One little trick is the use of the guard flag on the left hand side of the AND and the Throw on the right. This relies upon the fact that the compiler does short-circuit evaluation of logical conditions. In other words the compiler guarantees that if it knows the result of a logical expression simply by evaluating the left hand side then it will *not* evaluate the right. So if query is zero the Throw will not be done.

Next is the LOOP that operates the "Retry the delete?" message if the first attempt at deleting failed. Then the position of the record to be deleted is taken and is TryFetched. This is because the record needs to be full and accurate to allow the child links to be found and I cannot assume someone has made a record accurate just to delete it. Between the position and TryFetch is a block inside an IF SELF.UseLogout. This code is a horribly complex way of doing a simple thing. LogoutDelete (documented in part two of this article) simply finds out which files *may* be altered by this delete and adds them to the transaction frame.

Following this code is the main loop, which steps through all the relations calling DeleteSecondary for all files which are related with some form of constraint on the delete. (In C5 the LocalAction function filters out the RI done upon the server which does not require assistance from ABC). Note that DeleteSecondary is a method in the *related* RelationManager. This is a vital point! You do *not* go around deleting other RelationManagers' records; you ask them to do it for you. What gets passed in is the key of the His that this RelationManager is related to, the FieldClass containing the list of linking fields, and the action mode to say whether restriction, cascading or deleting is called for.

How does this function work? From the perspective of the current RelationManager, the answer is "Don't know, not my problem," but it *does* matter that I know *if* it worked. If it didn't I must stop processing myself. Note the little CheckError routine calls are pernicious: they can cause the whole method to be aborted. This code assumes the DeleteSecondary will have issued the ROLLBACK if required.

Assuming the children were OK then the RelationManager deletes its own record and handle any errors (including transaction rollbacks of child deletes if required).

```
Update PROCEDURE(BYTE FromForm=0),BYTE,VIRTUAL,PROC
```

The update code is very similar to the delete code so I'll focus on the differences. There is no need for the "Are you sure?" query. There's also no need for the `Position/Reget` as the code can assume someone doing an update has valid records in the buffer! Because updates cannot be restricted it's okay to update the primary record before cascading to the children. Again any errors are handled.

NAME="Update"The real interest (and new code) comes in the secondary loop. Note the call to `EqualLeftBuffer`. When an update is commenced in a form the `RelationManager's Save` method is called which snapshots all of the values of the linking fields of the relations into the `Buffer` portion of the linking fields `BufferPairsClass`. Thus at the update it's possible to compare the left (primary) record with those stored values. If they haven't changed (even if the record has) then there isn't anything to cascade.

Suppose the cascade fails. Now there's a primary record (in memory, the disk image will have been rolled-back) with linking fields that now *don't* point to the children. Yuk! So upon failure the code copies the linking fields from the child back into the parent to tie the records together again.

Halfway There

This ends the discussion of the methods that are clearly and logically related to each other. There are a number of methods that don't fall into as clear a classification, and I'll cover those next month in the second part of the `RelationManager` discussion.

Vol 1, No 8
Sept 1999

Main Page
Log In
Subscribe
Open Source
Links
Mailing Lists
Advertising
Submissions
Contact Us
Site Index
ClarionMag FAQ
Download PDFs
Search ClarionMag

The Clarion Advisor

Modifying Your Redirection Files

by Patrick K. O'Brien

I recently settled on a redirection file strategy that I rather like, so I thought I would share it here, for anyone who might find it useful.

Clarion uses redirection files for two reasons: to search for files when a path has not been specified, and to decide where to put any files it creates. The default redirection file (i.e., CLARION5.RED) is in your Clarion \BIN directory, but you can place a copy of the redirection file in your application's directory, if you want to make changes that affect only that application.

It took me a while to understand that all you really need in Clarion is an .APP file and a .DCT file. All the other files in your project get generated for you and can be deleted once in a while to clean things up or, on occasion, to solve problems. So it makes sense to separate these files into their own folder. At first I figured out what the file extensions were and started adding a line in the redirection file for each type of file that I wanted to separate, like the following:

```
[Common]
*.inc = .\src
*.clw = .\src
*.map = .\src
*.shp = .\src
...
```

This had the effect of creating an \SRC folder below the folder in which my .APP and .DCT files were stored. (Clarion creates this folder automatically, if it doesn't already exist.) Into this folder Clarion generated the files with the extensions listed. But Clarion generates a wide variety of files, and this was getting out of hand. I reached my limit trying to get the shortcut to the .EXE to go into a separate folder.

Then I decided to change my strategy a little. Now, instead of saying what to put where, I tell Clarion to put *everything* in \SRC, except what I *don't* want to go there. How do I do that? Well, here is what my Common section looks like now:

```
[Common]
*.exe = .
*.dll = .;%ROOT%\bin
*.tp? = %ROOT%\template
*.trf = %ROOT%\template
*.txs = %ROOT%\template
*.stt = %ROOT%\template
*.*   = .\src; .; %ROOT%\examples; %ROOT%\libsrc; %↵
       ROOT%\images; %ROOT%\template; %ROOT%\convsrc
*.lib = %ROOT%\lib
*.obj = %ROOT%\lib
*.res = %ROOT%\lib
```

Only two little differences exist between my version here and the default that is shipped with Clarion. The first change I made was to the *.* line. The original version listed the first directory as just a dot. This meant that all files created by Clarion would be put in the same directory as the .APP file. (Remember that only the first directory listed specifies where files *created* by the application generator should go.) I added .\SRC to the front of the line, which means, "Put everything in a directory called 'SRC' just below the current directory, and create this directory if it doesn't exist."

The other difference is the one exception I made. I wanted the final .EXE to remain in the same directory as the .APP file, so I added the line:

```
*.exe = .
```

If you wanted to keep the ship list file there as well, you could add the line:

```
*.shp = .
```

And so forth. Now all the files that Clarion generates go where I want them to go. And, if I start running into weird bugs, I can delete everything in \SRC and recompile. If I want to see the actual code that Clarion generates, perhaps in a different editor, all the .CLW and .INC files are in the \SRC folder. Try it. You might like it.

Do you have an improvement to, or variation on, this approach? Send it to advisor@clarionmag.com.

Patrick O'Brien is a partner with Orbtech, specializing in software development and database design.

Vol 1, No 8
Sept 1999

Feature Article

Survey Says...

by Dave Harms

Clarion Magazine recently polled its readers on which technologies they considered important, and which they were planning on implementing. Many of the technologies were suggested by newsgroup participants, and there were 127 survey responses. Not all respondents answered all questions, so on each question the number of responses is shown in parentheses.

I would like to say that this survey accurately reflects the Clarion community, but of course that's almost impossible to guarantee. The respondents to this poll all had access to the internet, and most would have learned of the poll through the newsgroups or through notifications to magazine subscribers (the survey was open to non-subscribers also). It may be that restricting the poll to developers on the internet skews the results somewhat, in particular when it comes to internet-related technologies.

As well, any survey is limited in its accuracy. You've probably heard the caveats on political polls: "This poll is accurate to within x percentage points 19 times out of 20." Something similar applies here, though I don't have any figures on the potential inaccuracy of these figures. I'll just say these numbers are probably indicative of something, unless they aren't.

So much for the disclaimers. What follows are some comments on the survey results.

An Aging Population?

The survey asked several general questions before getting on with the list of technologies. I was a bit surprised to see that over three quarters of respondents had been using Clarion for more than five years. On the one hand this should be a concern to Topspeed's marketing division. On the other hand the company clearly enjoys a lot of loyalty.

Years Experience With Clarion (127)

Under 1 year	1 to 3 years	3 to 5 years	Over 5 years
3%	6%	13%	78%

Most developers have also moved to Clarion5. The numbers on the DOS product are probably a bit misleading however as Clarion Magazine doesn't cater to DOS development, and to date hasn't carried any CDD or CDP articles.

Version Currently Used (126)

Clarion 5	Clarion 4	CW 2.x	CDD	CPD
96%	2%	2%	0%	0%

Although most developers have moved to Clarion5, a sizeable percentage still use the legacy templates. I'm also surprised to see that 6% are hand coders. It would be interesting to know if any of these people are using ABC in hand code.

Development Method (127)

Legacy templates	ABC Templates	Hand Code
24%	70%	6%

In keeping with the high average number of years with the product, the vast majority of respondents feel they have a good overall knowledge of Clarion, with a fourth considering themselves expert in many areas.

Clarion Skill Level (127)

Beginner	Familiar with the basics	Good overall knowledge	Expert in some areas	Expert in many areas
1%	4%	37%	35%	24%

So much for the background of the respondents.

The results of the technology questions are listed in the order they were asked, and are not ranked. If you want to get a quick idea of which technologies are the most important, pay attention to the two center columns: Importance=Essential and Implementation Timeline=Done. By this measure the top technologies among respondents are ODBC, SQL, internet protocols, and object-oriented design methodologies.

	Importance				Implementation Timeline					
	None	Some	Very	Essential	Done	0-3 Months	3-6 Months	6-12 Months	>12 Months	No Plans
SQL (125)	4%	12%	24%	60%	42%	16%	13%	10%	6%	8%
ODBC (119)	5%	21%	31%	43%	60%	9%	5%	5%	2%	13%
OLEDB (110)	15%	32%	29%	25%	5%	15%	13%	11%	5%	41%
Microsoft Transaction Server (105)	26%	30%	28%	17%	2%	4%	14%	13%	10%	48%
Object Databases (106)	20%	42%	26%	11%	2%	3%	5%	11%	18%	53%
ActiveX (116)	9%	17%	25%	48%	41%	13%	9%	8%	2%	17%
COM/OLE (117)	3%	19%	30%	49%	30%	12%	11%	14%	4%	15%
DCOM (108)	8%	25%	28%	39%	3%	9%	11%	19%	6%	39%
CORBA (104)	22%	40%	22%	15%	2%	2%	10%	9%	6%	63%
Java Beans (106)	25%	48%	15%	11%	2%	0%	8%	9%	8%	59%
WindowsCE (107)	24%	43%	21%	12%	2%	4%	3%	11%	10%	58%
Palm Pilot (106)	30%	42%	22%	7%	4%	2%	2%	10%	9%	63%
Embedded Systems (97)	46%	36%	14%	3%	4%	2%	0%	5%	8%	69%
Linxu (108)	12%	30%	39%	19%	10%	6%	6%	9%	16%	42%

Java (111)	14%	35%	32%	20%	11%	5%	11%	10%	12%	40%
Clarion Web Edition (115)	11%	28%	31%	30%	22%	12%	11%	18%	5%	24%
Internet Protocols (117)	3%	9%	27%	61%	39%	12%	9%	12%	3%	14%
XML (105)	9%	26%	34%	31%	7%	6%	15%	19%	9%	30%
DHTML (103)	13%	30%	30%	27%	15%	4%	7%	14%	9%	34%
ICMP (86)	31%	37%	21%	10%	2%	3%	3%	5%	7%	64%
TCP/IP and Sockets (108)	4%	17%	25%	55%	24%	19%	7%	10%	7%	17%
MS Remote Access Services (98)	20%	40%	26%	14%	16%	5%	5%	5%	5%	44%
Winsock (96)	14%	23%	28%	35%	27%	7%	4%	6%	4%	33%
MS Distributed interNetwork Ar (85)	32%	44%	15%	9%	2%	1%	2%	7%	7%	67%
OO Design Methodologies (117)	3%	12%	26%	60%	50%	9%	5%	9%	3%	15%
Unified Modeling Language (95)	16%	37%	29%	18%	13%	2%	3%	12%	11%	46%
Wizatrons (112)	17%	39%	26%	18%	18%	21%	15%	8%	2%	28%
Multi-tier architecture (97)	7%	23%	31%	39%	18%	8%	9%	13%	7%	32%
True OS Threads (98)	7%	22%	35%	36%	16%	8%	6%	7%	4%	43%
Microsoft Repository (87)	38%	45%	13%	5%	0%	0%	3%	2%	2%	77%
Voice Recognition (98)	33%	34%	23%	10%	5%	2%	4%	5%	13%	58%
DirectX (93)	33%	40%	17%	10%	3%	2%	3%	4%	3%	68%
Artificial Intelligence (93)	43%	33%	16%	8%	8%	0%	1%	1%	5%	69%
Novel 4.x 5 NDS Access (88)	48%	23%	22%	8%	7%	3%	2%	8%	1%	65%
Microsoft Message Queue (83)	28%	33%	24%	16%	4%	1%	5%	11%	4%	64%

In This Issue

Vol 1, No 8
Sept 1999

Feature Article

Industry Trends: How Important *Is* Java?

by Dave Harms

Clarion Magazine recently surveyed readers on the technologies they considered most important. High on the list were component technologies such as ActiveX and COM/DCOM, with a somewhat smaller percentage of readers showing interest in Java and JavaBeans. Java does come up for discussion occasionally in the Topspeed newsgroups, but on the whole I think it's a technology that's underrated by Clarion developers.

When Sun Microsystems introduced the Java language in November of 1995, it was with hopes that this new object-oriented language would revolutionize the programming, internet, and network computing worlds. The product was hot; the hype was massive.

It's fair to say that Java 1.0 didn't quite fulfill its promise, although it's hard to imagine any first version of a product living up to such optimistic press releases. But while the media frenzy waned, development continued. Over time Java (now in release 2) has gained a solid foothold in software development, though not in quite the way many pundits once predicted.

What does all of this mean to Clarion programmers? It means that Java is now one of the premier software development technologies (it's really much more than just a language, as the Java specification includes the runtime environment and a growing number of APIs). As such Java bears watching as much as Inprise's Delphi or any of Microsoft's stable of development products.

A Taste Of Java

My own interest in Java dates to 1995 when I began co-authoring the book "Web Site Programming with Java", published in 1996 by McGraw-Hill. (No, don't run out and buy it. This book is now quite dated.) I became involved in that project by virtue of having co-written "Developing Clarion For Windows Applications" with Ross Santos (SAMS, 1994), and definitely not because I had any special technical qualifications as a Java expert. That responsibility fell to my co-authors, both of whom worked for Sun. Writing and researching that book was an intense learning experience and my first significant involvement with an object-oriented (OO) language.

I found a lot to like in Java; for the first time OOP really made a lot of sense to me. And I'm convinced that if Clarion hadn't gone OO about the same time, I'd have long since left Clarion for Java. Happily, that wasn't necessary.

In many ways Java reminds me of Clarion, and vice versa. Both are powerful languages that put an emphasis on ease of use. Both are quite readable. Neither supports the use of pointers (well, Clarion sort of does), and both do a certain amount of automatic memory management, two key features that make a programmer's job much easier. There are many differences as well, of course. Java has a decidedly C-like syntax, is case-sensitive, and is designed from the ground up to be multi-threading, among others. As well Java is a "pure" OO language whereas Clarion mixes procedural and OO code. On the whole, however, I would classify Java as a high-productivity 3GL like the Clarion language (and here I mean just the language, not the Dictionary Editor and Application Generator and all that other good stuff).

It's Not What You Think

It's both fortunate and unfortunate that Sun launched Java as an internet development tool. The up side is that this gave the language unprecedented exposure. I recall seeing a number of prime-time news stories about Java during those early days. I mean news stories about a computer *language*! It was astonishing.

The down side is that Java is ingrained in the public consciousness as an internet-related product. This is almost entirely misleading.

Java Is Not JavaScript

The most discouraging result of Java's early hype is that many people equate Java with JavaScript. The SAMS book "Teach Yourself Java 2 Platform In 21 Days" sums up the relationship between these two products succinctly: "They have the same first four letters."

JavaScript was originally called LiveScript, and was created as a web page scripting language by Netscape. When Sun and Netscape entered into a marketing agreement which renamed LiveScript to JavaScript there was a lot of hand wringing among Java developers who (quite rightly) feared that Java would be tainted by association. Every time JavaScript fails on a web page it's the word Java that leaps out at the viewer. Which leads me to my second point.

Java Isn't Just Applets

Java isn't really a client-side technology. It can be used as a client-side technology, and again the early hype about Java was all about applets, little bundles of Java code that you could download and run on your web browser. But that's just a small part of what Java does.

Applets haven't proliferated quite the way most of us expected they would. Quite a few sites do use them, but in general internet bandwidth still isn't up to the task. This is one of the difficulties facing Clarion Web Edition, which uses Java applets to deliver Clarion screens to the browser. In some situations the Java applet solution is just too slow. And in any case, applets are really just a small part of the Java picture.

If you're like most people, almost everything you know about Java is wrong.

What's It Good For?

If Java isn't JavaScript, and applets aren't all that useful, then what's left? Quite a lot, as it turns out, both for web-related and "traditional" software development.

On the web front, Java is used increasingly for server-side development. Instead of applets, think of servlets, little bundles of Java code designed to run on the server. This shift began not that long after the release of Java 1.0, and was fueled by insufficient bandwidth and Java virtual machine compatibility problems. You can use Java to create web content on the client browser, but it's often more efficient (and safer) to do this on the server side and then deliver the finished product to the user as straight HTML, or HTML with a few bells and whistles. On many sites this kind of work is done in Perl, which is a widely used, robust, and almost completely bewildering script language. And although Perl is well-suited to this kind of work, it is by no means a general-purpose programming language. You wouldn't want to build a standalone business application with Perl.

Real Programming

Java is a real programming language. It isn't a scripting toy, and it isn't special-purpose, despite all that early press about how applets were going to save the world. You can use Java to write just about anything you want, excepting low-level code that deals directly with the hardware. Java exists at slightly higher levels of abstraction than many other languages, and that's a consequence of the attempt to make Java platform-independent. And this raises a number of concerns.

You've probably heard that Java is slow, and that it isn't a true compiled language. Both these statements are sort of true, and the reason is something called the Java Virtual Machine, or JVM.

If you're going to create a programming language that lets you write programs that can run unaltered on a variety of platforms then you're going to have to somehow reconcile the differences between those platforms (i.e. between a Windows PC and a Mac).

The Java approach is to not write software for a particular hardware platform at all. Instead, you write software to run on this other piece of software that sits between the Java program and the target platform. That piece of software is the JVM. There are JVMs for PCs running Windows, for the Mac, for Solaris, for Linux, and for a whole bunch of other platforms. The idea is that it's a lot more efficient to create tens or hundreds of JVMs than it is to create a multitude of versions of actual programs. The concept is good, but the problem is that you can end up having to write to the lowest common denominator.

Java does have mechanisms for calling platform-specific code, although doing this essentially defeats the "write once, run anywhere" purpose of using Java. That doesn't mean that it's a waste of time using Java, however, since programmers are generally more productive using Java than they are using C++.

The other major concern over the JVM model involves the speed of the executing code. Java compilers create Java bytecode, which is more like pseudocode than object code. The JVM executes the bytecode, but as this is a kind of interpretation there's some overhead involved. JVMs are getting faster, and speed concerns can also be addressed with Just-In-Time (JIT) compilers which translate often-used code to native code on the fly.

At present, Java code typically runs a bit slower than comparable C++ code, but for many applications it's an acceptable price to pay for the increase in productivity and the ease of maintaining Java code over C++ code.

The Java APIs and Components

At the same time as Sun has been refining the Java language, it has also been developing a number of APIs for everything from sound and graphics to e-commerce and telephony. These APIs have generally arrived later and less complete than most developers would like, but the lineup is starting to look fairly robust. Java also has a component framework called JavaBeans which competes with Microsoft's ActiveX/COM technology.

The component front in particular is heating up because these days it's not enough to use components within an application – you now need to be able to use components across applications, across intranets, and even across the internet.

In order to use a component that's somewhere other than inside your application you need some way to communicate with it, identify it, determine what capabilities it offers, send the component information, and get information back. There are, as you might expect, two primary competing standards for doing this.

CORBA And DCOM

One of these standards is Microsoft’s Distributed Component Object Model, or DCOM. DCOM is (more or less) specific to the Windows platform, and is Microsoft’s best hope for dominating the middleware market. CORBA is the collaborative effort of a bunch of people who are not Microsoft, and is governed by the Object Management Group. OMG was founded in 1989 by a group of eleven companies, one of which was Sun Microsystems.

Neither of these technologies is exactly new, and both have their strong points. DCOM certainly has the edge on the Windows platform, while CORBA has a "truer" object model and a greater measure of platform independence. This is of course an almost obscene oversimplification.

Of late, Java and CORBA have been getting significant support from corporate software developers. Although Java is owned by Sun, the specification for the JVM is publicly available. Java may not be as free or open as some would like (Sun still largely controls the language’s direction and has resisted most efforts to have Java’s future put in the hands of an independent standards body) but the relative openness of the specification means that any number of vendors can provide development environments.

Java and CORBA are headed for a major clash with DCOM and Microsoft development tools. A year or two ago the momentum was clearly with Microsoft, but that may not be the case any longer. Java and CORBA have both matured significantly, and perhaps spurred on by the DOJ action against Microsoft and the growing acceptance of open source software and the Linux operating system, many corporate shops are looking at solutions they perceive as less proprietary.

Will Java and CORBA win over Microsoft and DCOM? It’s never a good idea to ignore the 800 pound gorilla, and in any case it’s unlikely there will be a clear winner in the near future. More probably there will be some interoperability between the two systems.

What this does mean for Clarion developers is that in the future you’re likely to be hearing as much about Java, JavaBeans and CORBA as you do now about ActiveX, COM and DCOM. How this will affect our day-to-day software development remains to be seen.

Related Links

Java Home Page	http://java.sun.com
JavaWorld Magazine	http://www.javaworld.com
Gamelan – The official directory for Java	http://www.gamelan.com
Java Pro Magazine	http://www.java-pro.com
JARS Java Review Service	http://www.jars.com
Java Report Online	http://www.javareport.com

David Harms is an independent software developer and the co-author with Ross Santos of Developing Clarion for Windows Applications, *published by SAMS (1995). His company, CoveComm Inc, publishes Clarion Magazine.*

Vol 1, No 8
Sept 1999

Internet Framework Templates 2.0, Server Edition from Logic Central

Reviewed by Tom Hebenstreit

Review Editor's note:This is the first of two reviews of the Internet Framework Templates. This one covers the Server templates, while the second one will cover the Client Edition and how the two pieces fit together. Note that the Client edition is not required to use the Server Edition.

Logic Central's Internet Framework Templates are a relative newcomer on the Clarion Third Party scene, but they have already generated a fair number of questions on the TopSpeed newsgroups. The easiest way for me to introduce you to them is probably to begin by answering the most common of those questions. I'll also define a few terms that I'll use throughout the review.

(Acronym alert! It is impossible to talk about Internet communications without quickly becoming surrounded by the various terms and abbreviations that constitute 'Internet speak'. I will try to provide enough *simple* explanations and background so that the review is understandable, but it is far beyond the scope of this review to explain the bare metal details of the Internet.)

What are the Internet Framework Templates (IFT)?

These templates handle low-level communications across the Internet using the Windows Sockets (WinSock) libraries that are either built into or available for all versions of Windows. The protocol IFT uses for communicating over those sockets is HTTP (Hyper Text Transport Protocol), the same protocol that is the basis of the World Wide Web.

What is a socket (port)?

Sockets are tough to explain! Think of an old-fashioned telephone switchboard, where an operator answers all calls and then plugs each incoming call into a private connection with a local telephone. The socket is a bit like the operator. There is a single port (connection point) within an IP address that, by Internet standards, is a public port. For web servers, it is port number 80. When you send a request to a web server, that request goes to the public port, which then assigns a private socket/port (connection) with which you and the server to hold your conversation (in other words, request and receive data).

Once the server has assigned the private connection, it can move on to the next public request. In the meantime, you and the server use the private port until you are done, and then the port is freed so that it can be used again for someone else (just like hanging up the phone frees up a phone circuit). Note that you can have web servers (such as an IFT program) that listen to ports other than 80 (80 is simply the default). If you already have a web server on a particular machine, you can, for example, have an IFT server listening on port 81 or 89 or just about any other valid port number (the most common alternate address is 8080, which is easy to remember). To address a request to the non-standard port, you simply add the port number to the URL or IP address using a colon, as follows (this is a real link to Logic Central's IFT demo server):

http://24.5.80.133:89

In this example the HTTP request will go to port 89 rather than 80. If there is a server on that port, it will do exactly what a regular server does, create a private connection for the request and then wait for the next request on its public port.

What web servers does IFT work with?

Programs created with IFT do not work *with* a web server, they *are* a web server. This is a key point! You do not need a standard web server at all, as do other approaches such as CGI, Clarion Internet Connect, the Tornado OCX or many others. If you do happen to want to run an IFT program (or many of them) on a machine that has a standard server, you simply need to assign each one to different port.

What can I do with IFT?

The most common use of IFT would be to build what is called an "'application server." This is a specialized web server (program) that is built to provide one particular set of services, just like most of the Windows applications that you normally build with Clarion. The app could serve up standard web pages, custom pages built on the fly, send/receive data, send email (if you write code to do that), handle file uploads or just about any other service that is common across the Internet. In a way, you can think of an application server as a specialized form of client-server application built to operate over the Internet. The client requests something from your program, and your IFT program sends it back as either a web page or as binary data. If I sound rather vague about what exactly an IFT program can do, it is because you can use IFT to do just about anything you can imagine.

Ok, enough background.

Major Features

- The Internet Framework Templates handle all of the lowest level communications details, leaving you to attend to the task of handling requests and providing any and all *content*.
- The product is completely template code – no libraries or DLLs are needed (other than the standard Winsock DLLs).
- All standard methods for requesting data from a web server (GET, POST and HEADER) are supported.
- Additional utility functions are provided to take care of many common tasks, such as decoding information from forms and so forth.
- Bonus templates (called PowerMerge templates) are included to provide a variation of mail merge. This allows you to place tokens within other text and use these functions to replace the tokens with actual data. Thus you can write generic code or web pages, which are then filled in by the merge template code on demand.

Installation

The IFT package is provided as a standard single file install.

Installation proceeded without problems, but also without frills. The installer didn't locate my Clarion5 folder (it is on an E drive), didn't show any show any documentation and didn't register the templates for me. No icons were created and my Windows menus were not modified. The biggest shortcoming, though, was that it didn't tell me *where* IFT was being installed. I had to poke around in my Clarion5 directory to discover a new folder named \IFTEXAMPLES.

Once I located that directory, I found both the documentation and two example applications.

Documentation

I normally leave discussions of documentation until the latter half of a review, but in this case it needs to be mentioned right up front. The documentation is, in a word, sparse, totaling about six or seven printed pages. The biggest pointer in the docs is to study the sample applications that are provided, as they demonstrate most of the common techniques that you would use to build fully functional IFT servers. Two demo apps come with IFT, one to illustrate how to use IFT, the other to show how to use the PowerMerge templates (they are used in the IFT demo app as well). On the minus side, the demo apps are not very well commented, so you will need to study them fairly closely.

Additionally, since the templates provide a framework for other technologies such as HTTP and HTML, there is a definite learning curve beyond that of the templates themselves.

So, be forewarned – while the templates themselves are very simple to use, you will need to be prepared to spend some quality time tracking through the code they generate in order to learn just where your custom code should go. You will also really need to learn some basics regarding HTTP, HTML and the theory of how these things all work together. That information is not covered in the documentation either, but there must be a *gazillion* books on HTML out there, many of which you can pick up cheap in the remainder bins for $5-10. On top of that, there is no better place for learning about web technologies than on the web itself.

Time to move on to the templates themselves.

Implementation

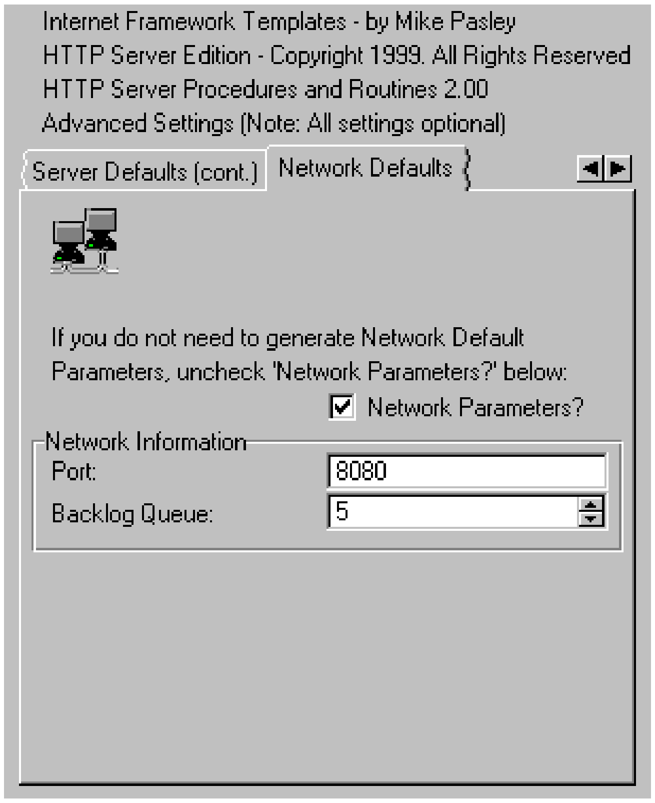
Starting at the beginning, I registered the five templates that comprise IFT (two for IFT itself, the other three for PowerMerge and some demo functions).

I then created a server using Clarion 5 according to the steps laid out in the documentation. Basically, this involved:

1. Creating a new 32-bit application (ABC templates)
2. Adding the IFT Global Extension template
3. Creating a simple window procedure, and adding the IFT Start/Stop buttons control template to it.
4. Adding the IFT Procedures and Routines extension template to the window.
5. Compiling and running the application.

Pressing the ‘Start’ button on my server window (as shown below in Figure 2) started the actual process of having my server listen for requests on the selected Port (Figure 1 shows how I specified the port in the templates).

Figure 1. Specifying the server port.



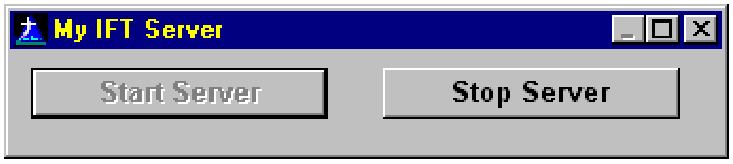
Internet Framework Templates - by Mike Pasley
HTTP Server Edition - Copyright 1999. All Rights Reserved
HTTP Server Procedures and Routines 2.00
Advanced Settings (Note: All settings optional)

If you do not need to generate Network Default Parameters, uncheck 'Network Parameters?' below:

[✓] Network Parameters?

Network Information
Port: 8080
Backlog Queue: 5

Figure 2. My test server in action (not much to look it, is it?)



Once it was up and running, the next step was to test it out by sending it a request from a web browser. I fired up Netscape 4.61 and typed in the following URL (this is *not* a real world wide web address, though it is valid on my test machine): http://localhost:8080/test

Lo and behold, up popped a page with a "Test Successful!" message. Where did it come from? The templates offer an option to generate a procedure that responds to the /test page with a user definable message. This is a nice touch, as it lets you verify that things are working without doing anything other than the above five or six steps. Remember, if I hadn't specified the port (8080), my request would have gone to my normal web server (which is running on the same machine) rather than the IFT server.

Well, that was certainly easy. The problem, though, was that my spiffy new IFT server didn't do anything useful yet. Well, that's not really true. By making its default folder a directory on my machine that contained an actual web site, it quite happily served up pages and images from that site. Neat! Right out of the box I had a basic web server built in Clarion.

Moving along, I compiled and ran the demo application. Whoops. Trying to request pages from that IFT server didn't work at all. I received either strange messages or Netscape (and IE) would try to download zero byte length pages to my hard disk.

After a bit of research I discovered that IFT had defaulted to C:\Clarion5 for the demo home directory. Once I changed that to the real location on my E drive, and stopped and restarted IFT, the demo started working. Well, sort of. I tried the /test parameter like the first time and it didn't work. Back to the app for another look and I discovered that the demo was set to respond to /mytest rather than /test. This is another case where the documentation should be clearer about what is needed to run the demo, and it seems to me that it should be consistent with the documented /test syntax as well.

A key point to understand when working with IFT is that the URL (page) you request may or may not be a physical file on disk. For example, when the IFT server received the request for /test, it already knew that this was a special case. Instead of looking for a file to send back, it simply created the response right then and there, in memory, and then sent that response back to the browser case (see Listings 1 and 2 below).

It may *look* like you are requesting a physical file, but in reality you are simply passing a parameter to the IFT server (just like calling a parameterized procedure in Clarion). In cases where there is no special code for handling a request, IFT automatically looks on disk to see if there actually is a file and then sends it back if the file is found. That's why the /test request worked even though the demo directory was set wrong (it never looked for a directory), and also why IFT works as a normal web page server right out of the box.

What came next? Well, to be frank, it was a number of days wading through both the generated code and the demo code, studying how it handled the various kinds of requests that can be submitted via HTTP, asking questions, copying code to my applications and in general just getting a feel for the amount of flexibility the templates offer (a LOT, by the way). Fortunately, I'm already quite comfortable with HTTP and HTML, so at least I didn't have that mountain to climb.

To give you a bit of a feel what is involved, here is the code that is generated by the templates to send back that automatic test page. First, the bit that checks for \test as a special case. This is part of a larger section of code that parses out what the user requested, and then tests to see what that request was.

Listing 1. Checking for the test parameter.

```
 Case UPPER(mPathName)       !mPathName contains the request
of UPPER('test')
   OnSendTestData(ReqSocket)
! lots more ‘OF’ this or that tests
```

That's pretty straight forward. Here is the actual procedure to create the response page:

Listing 2. Creating and returning the Test response page

```
OnSendTestData PROCEDURE(SOCKET ReqSocket)

  Code
  clear(sendbuf,-1)
  clear(SendHDRbuf,-1)
  sendbuf='<<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">'
  sendbuf=sendbuf&'<<html><<head><<title>Test Message<</title>'
  sendbuf=sendbuf&'<</head><<body><<center><<font face="Verdana">'
  sendbuf=sendbuf&'<<H2><<font color="#FF0000"><<P><<P><<P><<P>↵
    Test Successful!<</font><</H2>'
  sendbuf=sendbuf&'<</font><</center><</body><</html>'
  SendHDRBuf='HTTP/1.0 200 OK<13><10>'
  SendHDRBuf=SendHDRbuf&'Date:'&DateTimeStamp(Today())&'<13><10>'
  SendHDRBuf=SendHDRbuf&'Server: IFTTester/1.0<13><10>'
  SendHDRBuf=SendHDRbuf&'MIME-version: 1.0<13><10>'
  SendHDRbuf=SendHDRbuf&'Pragma: no-cache<13><10>'
  SendHDRbuf=SendHDRbuf&'Content-type: text/html<13><10>'
  SendHDRBuf=SendHDRbuf&'Last-modified:'&DateTimeStamp(Today())&'<13><10>'
  SendHDRBuf=SendHDRbuf&'Content-length: '&len(sendbuf)&'<13><10><13><10>'
  sendbuf=clip(SendHDRBuf)&clip(sendbuf)
  Err=ssend(ReqSocket, sendbuf, len(sendbuf),0)
  OnCloseClientSocket(ReqSocket)
```

Now, before you run screaming from the review, that there are really only four things happening here.

1. The two text buffers that will hold the response page are cleared. Why two? If you look, it is building two things. The first is the actual page to be displayed in your browser, and that page contains standard HTML. The second is the HTTP header that describes the page so that your browser knows where it came from and how to display it. Most of the code is simply concatenating text into two big strings. Oh, yeah – and why is the header created *after* the page? If you'll look at the line 4 up from the bottom, you'll see that it is using the standard LEN() function to get the length of the HTML page. This becomes part of the header so that your browser knows just how many bytes should be read from its port (and that number isn't available until after the page string is complete).
2. The two buffers are concatenated. Here is where the HTTP header is placed in the proper position to be sent first.
3. The IFT `ssend()` function is used to actually send the information (the buffer) back to the requestor (you and your browser) using the private socket.
4. The socket is closed since we are finished with it.

That's not so bad now, is it? You can see though how having some knowledge of HTTP and HTML can be important (how else will you know what to send back?).

Be aware that you *can* use visual design tools such as Microsoft FrontPage, Netscape Composer, Macromedia Dreamweaver and so on to create your pages. You can then use the included PowerMerge templates to merge data into those pages where appropriate. You are *not* forced into hand coding all headers and HTML.

Performance

Once I actually got an IFT-based server running, the performance of my IFT created server programs was very good. Remember, when compared to something like IIS (Microsoft's web server - Internet Information Server), an IFT program is a lean, mean server machine unencumbered by the need to do anything beyond what *you* tell it to.

> Note: I will be doing more performance testing over the next few weeks as I begin working with the Client version of IFT, and will include more performance information in that review.

Technical Support

Technical support was excellent. I bombarded Mike Pasley (the author of the templates) with an endless stream of email questions, problems (mostly my own) and comments. All responses were fast and informative. One template problem I discovered in the PowerMerge templates was fixed by the next day, with a new version waiting in my mailbox. Another problem was solved within an hour so.

Logic Central also monitors the TopSpeed Third Party news group. Questions posted there are generally answered in short order.

I must point out, though, that Logic Central's Technical Support is just that – support for technical problems related to the templates. They can't be expected to teach you the ins and outs of HTTP, HTML, URL encoding/decoding and all those other nuts and bolts that can be used to build a web-based application server (remember that learning curve I talked about earlier?).

Room for improvement

As I'm sure you have discerned by now, IFT as a product is definitely in need of better documentation. Eminently useful templates and utilities such as the PowerMerge functions were not documented at *all* in the version that I was testing. Logic Central is aware of these shortcomings, though, and hopefully the situation will improve as they have time to work on it.

The documentation also suggests looking at the embeds to help learn templates. By my count, there were more than 65 IFT-specific embed points in my test server - yikes! Not that I am complaining at all about there being so many embed points; that's a very good thing and adds to the flexibility. I just don't want to have to manually inspect them all to learn the product!

The install could also be improved by making more basic information (such as the installation directory) available to the user.

Summary

The Internet Framework Templates are a very powerful tool which open up an entirely new avenue for using Clarion to build Internet-based applications. Ironically, the fact that IFT makes it incredibly easy to get close to the metal is both its greatest strength (indeed, its reason for existence) *and* its greatest drawback. After all, once you are there you *are* left close to the metal, and it's up to you to do the rest.

The bottom line is that if you have a need to build any kind of Internet application server, and are willing to put in the time to learn basic Internet standards and skills such as working with HTTP and HTML, you really can do virtually *anything* with these templates.

That *is* why they are called a Framework, after all.

PRODUCT RATING	
Overall	▲▲▲
Ability to do the task	Very Good
Ease of use	Fair
Ease of installation	Fair
Documentation	Fair
Technical support	Excellent
Black box DLLs/LIBs	No

LEGEND	
First class all the way	▲▲▲▲
More than adequate	▲▲▲
Barely adequate	▲▲
Don't even think about it	▲

The Internet Framework Templates Server Edition list for US$299, and can be purchased via the Internet from BMT Micro.

The IFT Server purchase page is: http://www.bmtmicro.com/catalog/ifthttpserver.html
The IFT Client purchase page is: http://www.bmtmicro.com/catalog/ifthttpclient.html

Special Subscription Offer
Posted on September 22, 1999

Technology Poll Results
Posted on September 21, 1999

Industry Trends: How Important Is Java?
Posted on September 21, 1999

Product Review: IFT Server
Posted on September 21, 1999

September News
Posted on September 21, 1999

If you are buying both at the same time or have bought one and you are returning later to buy another you can receive a $20 discount (after Logic Central has verified your prior purchase).

More information, demos and answers to frequently asked questions can be found at the Logic Central web site: http://www.logicentral.com

Vendor Comments from Mike Pasley of Logic Central

The IFT 2.01 version will be coming out with additional functionality including a Base64 codec and use of the WWW-Authenticate header. Documentation, as has been noted, will be increased and improved to respond to the growing user base, which is including many developers new to Internet programming. The Internet Framework toolset will continue to expand to make even new internet protocol developments more accessible to the Clarion developers. Technical support will come to include a live chat session (probably) at the Logic Central website. Please continue to view Logic Central and Clarion Magazine web sites for the latest products and news from Logic Central. The IFT will continue to expand its editions and their functionality. Suggestions will continue to be welcomed. I would also like to take this opportunity to thank all who have given me support on this project.

Vol 1, No 8
Sept, 1999

Main Page
Log In
Subscribe
Open Source
Links
Mailing Lists
Advertising
Submissions
Contact Us
Site Index
ClarionMag FAQ
Download PDFs
Search ClarionMag

Clarion News

September 21, 1999

DevCon '99 Special Subscription Offer
To celebrate DevCon '99 Clarion Magazine is offering a special deal to new subscribers: 20 months of ClarionMag for $99, a regular value of $125. For $99 you get all the back issues, all the issues with DevCon '99 coverage, and a year's subscription beginning November 1.

Multi-Proj Special Ends September 30
The Multi-Proj "product of the month" special offer ends September 30. This batch compiler is on sale for $49, $40 off the regular price.

ABCFree Templates and Tools Updated
New in the ABC Free Templates and Tools: global template to set browse property; global template to limit application to one instance.

More (Free) API Madness
Steve Bottomley has uploaded CLEARDESK.ZIP to the IceTips incoming directory. This program uses Windows API calls to set options for Desktop icons. Add to your Startup group and using two command line parameters, you can set options for icons on your desktop to display as large, small, or details (file details). You can also set the background of the text section to display as transparent (ideal if you use bitmaps as your background) or standard.

Das Tools Version 5.1.0 In Beta
Tinman Development has released Das Tools Version 5.1.0 for general beta testing. This version is for C5a and above only. Both template sets are supported. Included in this version are the Castle Quickeasy templates.

TimeSavers Scheduler Templates Pre-Release
The TimeSavers Scheduler Templates are 99% complete, and a new demo is available. Included: A calendar creator template that lets you dynamically create any kind of calendar you want; a scheduling grid template that lets you create monthly, weekly, or daily scheduling grids. All source is included. Introductory price is $99, or $79 through Wednesday, September 22, 1999.

Query Wizard 5 With SQL Syntax
Query Wizard has been updated with an option to create SQL syntax queries. This feature supports CCS SQL5 as well as the ABC template set. ABC support is rendered via PROP:SqlFilter. Other new features include enhancements to Expression Builder, a new control template to allow advanced users to directly enter queries, a dropdown calendar interface for date columns, and a variety of improvements. The upgrade from QW4 to QW5 is just $59 during the pre-release, $79 after the product begins shipping (in early October). Pre-release purchasers will receive the current beta with the functionality described above. Full price for Query Wizard 5 will be $229 at the final release.

Gitano DevCon Specials
Gitano DevCon specials are in effect from Sept 20th to Oct 2nd and include an across the board 15% discount, free polo shirt with orders

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

etc 2000
If you're interested take our poll & let us know!

over $200 (plus $5 shipping, US delivery only), and introductory pricing on GCalPro.

September 21, 1999

Stealth Software On The Road
Stealth Software will be on the road (in the air) until after DevCon and some emails may go unanswered until the first of October 1.

September 14, 1999

Hurricane Closes Topspeed Headquarters
Due to the impending arrival of Hurricane Floyd TopSpeed has closed its offices through Tuesday, September 14th. If you have difficulty connecting with tsnew.clarion.com you may access the server's mirror site at news.clarion.com.

Dalby Source Code Printer 5.0 Coming Soon
Dalby Source Printer 5.0, which prints formatted/syntax highlighted Clarion code, will be released soon as a commercial product. , Changes in version 5 include: fixes for Clarion 5; 32 bit program with support for long filenames; option to print only embed code; and improved embed headers. Cost: $80.

Creative Reporting Tools v5.15 Available
Creative reporting tools version 5.15 is now available for download from the CPCS web site. This new version is free to all registered users of v5.xx, 4.xx, and v2.25. (use your latest authorization codes to install). This version incorporates a new option on the Universal ABC Report template to allow use of ABC FileManager methods for files in the procedure's file tree.

LSPack Unpack Library Hotfix
Linder Software has a hotfix available for the LSPack Compression Library. This hotfix corrects a bug in the unpack LSP_U_FILERENAME function. Users of In Back (J&S Software) and BackFlash (Sterling Data) don't need to download this hotfix.

IFT Introductory Offer Extended
The Internet Framework Templates HTTP Server Edition introductory offer of $99 has been extended until September 20, 1999. Also available is Zap! Direct Messenger SS. It demonstrates combining the HTTP Server with the HTTP Client. The ZDM SS demo interface is courtesy of the upcoming IFT ShapeShift templates.

Whitemarsh Article Of The Month
The September 1999 Article of the Month from Whitemarsh Information Systems is about how to get valid requirements for a business information system's implementation. There are many examples, all done in Clarion. Topics include: the statistics and reasons for Information Systems failures; why tools such as Clarion can reduce costs by about 66%; a nine-step process to IT success; an overview of the Whitemarsh Information Systems Plan process; an overview of the Whitemarsh's approach to project management. A full set of overheads will be available in mid October.

Free Templates
Sergey Bashkiroff has published several free templates including a context help extension, adding DebugOut debugging information (for DbWin32), and a disk notify class.

September 7, 1999

CapeSoft Multi-Proj 2.0 Ships
CapeSoft has shipped version 2 of its Multi-Proj batch compiler, a tool for automating multi-DLL or multi-EXE compiles. Includes support for multiple versions of the same app, writes templates for sharing functions and data, and allows for file driver substitution.

Templates and Class For Controlling MS Word
SoftmasterS has a class with templates which provides control of MS Word within Clarion programs. Functions include showing/hiding Word,

creating documents, executing macros, text, font, and color control, moving the cursor, table creation, and more. A demo version is available.

ActiveX SMTP Mailer

The Network Connection Inc. has a new ActiveX SMTP Internet mailer written for Clarion. Available with or without the ability to do attachments ($49 and $29 respectively). Also at this site, an ADO ActiveX control.

New Clarion Source Web Ring

Gitano Software has formed a new Clarion source Web Ring, which allows web surfers to travel from site to site. Membership is open to anyone who has something of value to contribute to the Clarion community.

Handy Tools 'M' Build

The Clarion Handy Tools 'M' build is now available. New/changed in this build: Name and address controls that emulate the MS Outlook (not Outlook Express) approach; enhancements to the email classes; new demo program added to FTP template (shows how to use TimerDownload()); compile manager; file search.

Ragazzi Software Update

New/updated products from Ragazzi include: the Application Window Printer (corrected file selection procedure); simple thread manager template (allows only one instance of any procedure); Edit/View Manager which only allows users to view data based on a conditional expression; a browse list sequence template which lets you move items up and down in the list (requires an additional sequence field in the key); procedure standards global extension; pop-up calendar enhancements.

Clarion Source Search Goes Gold

Carl Barnes' Source Search utility is now in final release. This utility program is designed specifically to search Clarion Source Code, show you the line of source and tell you the module, procedure, method and routine where the results were found. Price is $45.99.

Read the August 1999 News

Read the October 1999 News

Do you have a news story or press release we should know about? Send it to editor@clarionmag.com

Vol 1, No 8
Sept 1999

DevCon '99 Special Report

DevCon Preview Part 1

by Dave Harms

Clarion Magazine has landed in Fort Lauderdale!

Well actually it was me and I landed at Miami International late Saturday afternoon. Tom Hebenstreit, Clarion Magazine's reviews editor is here as well. The magazine is still on the server, where it always is.

I don't have much to report about the flight, other than it took place on an airplane. It did take almost as long to retrieve my luggage in Miami as it did to travel from Winnipeg. My fellow passengers and I gathered at the baggage claim, where for a rather long time we stood and watched two lonely, lost pieces of luggage appear at one end of the conveyer and disappear at the other end, over and over again. Then a teddy bear appeared on the conveyer, face down. Was this an omen? Had our luggage been hijacked? Was the teddy bear a message from a desperate band of renegade toy salespeople? Evidently not: fifteen anxious minutes later I had my suitcase in hand.

My luggage and I exited to the rental car shuttle area. I don't know exactly how many rental car companies there are in this country but I'm pretty sure that every one of them sent its shuttle roaring past me before the one I wanted showed up. The exhaust fumes in the enclosed roadway were beginning to get to me. I was thinking positive thoughts about Wizatrons.

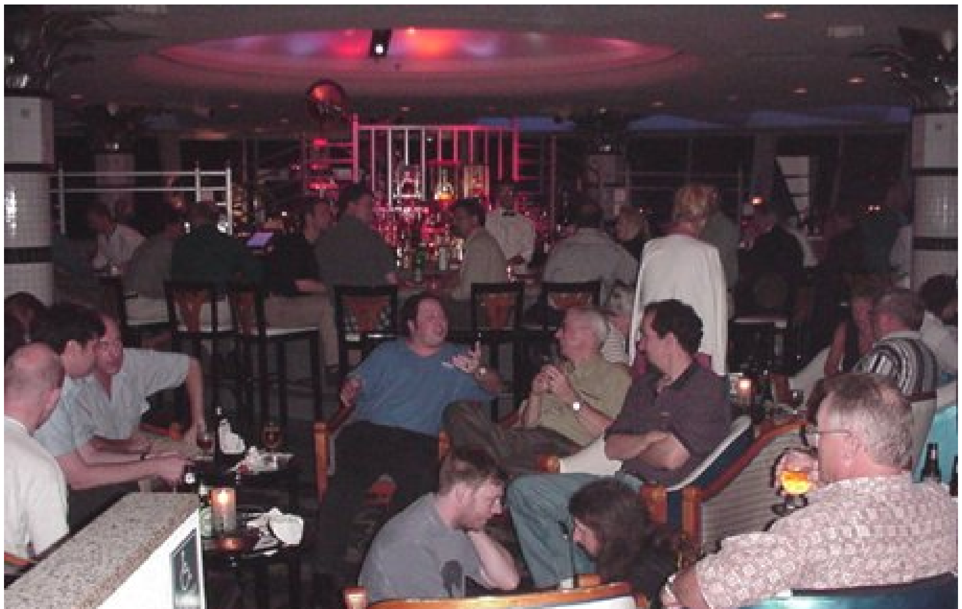
Happily the shuttle was carrying a load of fresh air and I had a chance to clear my head. On the other hand, once I had my rental car a clear head didn't seem to help me grasp the directions to the hotel. I made a few wrong turns. The instructions said take Davie Blvd east to US1, but neglected to say what to do when I got to US1. I turned south. No, that must be wrong. I went north. That was definitely wrong, so I went south again, then left. Okay, another left turn before the Intracoastal Waterway, said the instructions. Well it looked a whole lot like an Intracoastal Waterway to me, so I turned, just in time to see the sign telling me I should have stayed in the right lane over the bridge to get to the Hyatt.

Back on the bridge, and a huge lineup for the Hyatt. Hey, Clarion's hit the big time! Well maybe not just yet – there was some major event besides ours at the hotel. Finally I reached the lobby, got checked in and settled in, and headed out to see who else made it.

Turns out the guys were all up at the lounge at the top of the tower (wouldn't take a rocket scientist to figure that out).

Among those present, Andy Ireland, here shown on the bridge of his own personal starship.

The guy at the bottom right edge of the picture below with his back to the camera is Larry Teames. If he looks a bit put out it's probably because no one believed his fairly extensive treatise on the Fargo programming language. Hey, Larry, it's a town in North Dakota!



Also of interest in the above picture is the two guys on the floor, Bruce Johnson and Tom Hebenstreit. You may think they're in earnest conversation, or maybe just a bit wobbly and having trouble standing up.

In fact, as this photograph shows, they're practising for their DevCon seminar on yogic flying for Clarion developers. If you look closely you can see they're about half an inch off the floor.

Now Tom's taken a rest while Bruce hovers four feet off the ground, greatly impressing Carl Barnes. Unfortunately our photographer was caught off guard and didn't get a picture of Bruce in flight.

Sunday Morning

Registration started Sunday morning, and developers were out in force. You just can't tell it from this picture. I mean, you have to register developers one at a time anyway, right?

There were well over a hundred deveopers in the Sunday morning session, which was conducted by Russ Eggen (keep an eye out for the "I was Russ'd and survived" buttons).

At left, Gus Creces picks up his DevCon polo shirt and realizes he needs to put on some weight. This, ah, puts him in the minority.

The final word in this report goes to Topspeed's Richard Chapman, who in an

unguarded moment spills the beans about Topspeed's future technology direction.

MPEG Video (200k)

Vol 1, No 7
August 1999

CoveComm Inc

Clarion magazine

Press Release

DevCon '99 Special Offer

Get 20 Months Of Clarion Magazine For Just $99!

DevCon '99 is just around the corner and Clarion Magazine will be there to keep you informed on conference happenings. To celebrate DevCon '99 we're offering a special subscription rate of $99, which includes all the back issues, all the DevCon issues, and a year's subscription:

Back & DevCon Issues: Mar/1999 to Oct/1999	$50
One year subscription: Nov/1999 Oct/2000	$75
Regular Total Price	$125
DevCon '99 Special Offer	**$99**
YOU SAVE	**$26**

Click Here To Subscribe

This Offer Expires Midnight, October 8th, 1999.

If you wish to pay for the DevCon Special by check or money order, please send an email editor@clarionmag.com. If you want to see the regular subscription options click here.

What's In The Back Issues?

Here's a sampler of what's already appeared in Clarion Magazine:

The Novice's Corner articles have covered the basics of the Clarion development environment, understanding templates and embeds, designing databases, and many-to-many relationships. David Bayliss has written extensively on the internals of the ABC class library, to date covering ErrorClass, FieldClass, ConstantClass, FileManager (in parts one, two and three). David continues this series with the RelationManager. He has also written on Wizatrons and has done some interesting analysis of Clarion Challenge Results.

The ABCs of OOP series by Dave Harms (in parts one, two, and three to date) covers object-oriented programming principles and practices in the context of ABC. Steve Parker is a regular contributor to Clarion Magazine, and his works include a discussion of NAME() and a three part series on handling sort orders.

One regular feature is the Clarion Advisor, which offers advice on common programming problems. The Advisor has covered topics such as debugging using project files, Topspeed driver error codes, custom editor colors, fast ASCII file access, and more.

Clarion Magazine frequently interviews movers and shakers in the

In This Issue

Clarion software development world. Roy Rafalco spoke with Clarion Magazine shortly after his appointment as Topspeed's new CEO, and offered many insights into how Topseed functions and where the company is headed. You can also read about Ragnar Hellspong, creator of ForKeeps, and the three Clarion developers behind the internet's Clarion knowledgebases.

We're also fortunate to have as regular contributors Andy "Cowboy" Stapleton, who rides herd on a variety of SQL databases, and Larry Teames, one of the best-known third party vendors and an expert on reporting. You can find their articles, and a whole lot of other things, by going to the Clarion Magazine search engine. Search for, respectively, "Cowboy" and "Teames".

Clarion Magazine also contains product reviews (such as Datamatrix's Xplore templates and Sterling Data's Searchflash) and the ever popular Clarion Challenge series. Tom Hebenstreit has joined Clarion Magazine (from the now defunct ClarionOnline) and will be heading up product reviews.

For the truly hard core hand coder there are in-depth treatments like Jim Kane's assembler-driven technique for calling OLE methods. And for those who like to download and read at their leisure (leisure, what's that?), each issue of Clarion Magazine is also available in PDF format. PDFs are also recommended if you want to print an issue.

This is a sampling of what's been available in Clarion Magazine over the past six months. For a complete listing please see the site index.

What's Next?

In addition to DevCon '99 coverage, we have a lot of terrific articles waiting in the wings. The Novice's Corner series will be resuming shortly, David Bayliss continues with his explanation of the ABC class library, Tom Hebenstreit reviews the hottest new Clarion-related products, and a host of regular and new writers help you make the most of Clarion Development. Subscribe now and save over 25%!

Dave Harms
Publisher

Vol 1, No 8
Sept 1999

CoveComm Inc.

Clarion magazine

DevCon '99 Special Report

Day One Overview

by Dave Harms

Day one at the '99 DevCon has come and gone, and it's been interesting. As the promotional materials indicated, the focus really has been on the Internet (or more accurately, on web-enabling for the Internet and intranets. There have been some indications of TopSpeed's future direction on the necessary technologies, but pending Bob Zaunere's presentation Tuesday morning, that's pretty much all there is to go on.

Roy Rafalco gave a brief introduction to the conference and to some of the new people at TopSpeed. He also likened the challenges facing Clarion developers to a hurricane warning, an appropriate enough metaphor since DevCon is traditionally held during hurricane season in Florida. Roy also referred to recent news and some advice from the Gartner Group about a coming shift away from the Windows PC platform. According to the Gartner Group in particular, Windows software development targets are expected to drop from 60% to 40% in the near future. Java/CORBA development also outpaces COM development by a 2:1 margin. Indications are that while Microsoft dominates now, the field is about to open.

Roy Rafalco, Topspeed's president and CEO, started the conference off with a bang, or a series of bangs to be more precise. The chairs in the morning general session had helium balloons attached with strings, and inside some balloons were prize certificates. Watch the [MPEG video](#) (960k) to see what happened.

As Bruce Barrington said afterward, getting a group of Clarion developers to wait for a three count to pop balloons was like herding cats.

The keynote address was delivered by Hank Asher, founder of DataBase Technologies, a publicly traded company with a market capitalization of some $500 million that provides law enforcement database services. Hank's a big Clarion fan, and is behind eData, the company some of you may know for hiring Clarion's "best and brightest" to develop a new product. In the morning session, we all got a first-hand look at the result, as demonstrated by eData's Tom Moseley.

EData's product is a massive 20-30 terabyte database containing massive amounts of information on US businesses and individuals. The search engine is written in C and the client side product (which is pretty slick) is written in Clarion. Any user with the client software can locate data on, say, sales prospects, obtain the required results, and then request a report. A fee is charged, processed, and the report is made available for download.

Asher mentioned that Sun Microsystem's Scott McNealy had visited eData twice recently and was blown away by what eData had done.

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

etc 2000
If you're interested take our poll & let us know!

Partway through the presentation a fire alarm sounded. The siren lasted only a short time, but Asher was forced to continue for several minutes amid a loud beep every five seconds and a strobe flashing behind the presentation screen.

Following the keynote address Greg Gubrud of ONTOP Systems presented his company's work with the Internet Connect product, outlining in particular the work done to create Java-free (aka caffeine-free) versions of their applications. The company has also created IC apps that use Java but local ISP and broker problems prevented him from demonstrating these live. He had better luck with the caffeine-free apps and a long-distance connection to his own ISP.

Clarion Magazine will have more on the Monday sessions later this week, including photos, MPEG video, and an expanded look at the keynote address.

So far at least the conference has focused heavily on the Internet/Intranet web application server area. Both Roy Rafalco and Hank Asher repeatedly talked about the opportunities in this area, and the need for a massive (worldwide) rewrite of applications. The title of Esli Badenhorst's general session was "Web Enable Now! Or Crash And Burn!" Whether that sentiment is quite correct remains to be seen, but certainly there is a major shift happening.

How TopSpeed expects to address that shift isn't completely clear yet, although there is a new release of Internet Connect in the works. Tuesday morning Bob Zaunere will present that future direction.

Monday wasn't all Internet though. A few of the highlights were Sebastian Talamoni's fast-paced, flashy, and informative Wizatron overview, Joe van Niekerk's look at new features in ABC and the Clarion language (you're going to like "interfaces"), and Tom Hebenstreit's session on building distributed applications using email. More on those and other sessions to come.

Vol 1, No 8
Sept 1999

Main Page
Log In
Subscribe
Open Source
Links
Mailing Lists
Advertising
Submissions
Contact Us
Site Index
ClarionMag FAQ
Download PDFs
Search ClarionMag

DevCon '99 Special Report

TopSpeed's New Direction: The Web

by Dave Harms

9/28/99 1:02 PM

Monday's focus on web development has been confirmed by Bob Zaunere's presentation this morning: TopSpeed is making a major directional change toward deploying applications on the web.

The short term future for TopSpeed is Internet Connect 2, a major revision that replaces the Java controls with HTML controls, with no significant loss of functionality and a massive increase in speed. A beta of the product is expected to be released in October.

Jim DeFabia demonstrated a IC2 counterpart to the IC1 application he developed for the Monday session. The IC2 app used a table to display a browse of items, and provided locator and scrolling support. Although any change in the data requires a full refresh, the page size is small. Based on the demonstration, this really does make IC2 (aka the Web Edition) competitive in the world of web-based application. And deployment can be very fast as it allows Clarion developers to leverage their existing knowledge of the development environment. This is a big step forward for IC.

Bob Z also spoke about the product to come after IC, called iBuild. In contrast to IC, which generates HTML at runtime, iBuild will work with HTML created at design time. TS has entered into an agreement to OEM the full version of DreamWeaver to support the HTML design side of the process. Through the use of wizatrons, HTML changes will be reflected in the application, and vice versa. (Zaunere later said that C6 will probably be the Web Edition as TopSpeed doesn't expect to see a lot of developers doing Windows-only development in the future.)

Also part of the iBuild direction is a set of web application development kits which can be resold and/or modified by developers. Currently in the planning stage are iSell, a web storefront and store management system, iPublish for administering web content (including user contributions), and iMarket for tracking customer/contact management. These applications are designed so they can be integrated into a seamless whole on the web site.

There are a number of enhancements to the ABC library to help support all of this, as described by Joe van Niekerk in the morning's final session. These include classes for SMTP and MIME (email) support, which use the new Winsock class(es). All of this code also makes use of the new language support for interfaces, so integration with third part products should be expedited. Templates are included, and an alpha version should be available in about six weeks.

Overall reaction to the change in Topspeed's direction has been positive as most developers recognize the inevitability of the shift to the web.

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

etc 2000
If you're interested take our poll & let us know!

What's less clear is how the new product lines fit the kind of work many Clarion developers do. There are also some concerns over the pricing model which although not fully specified seems to indicate that developers would license products such as the web development kits on a site by site basis.

Site hosting for developers will be available through an agreement between TopSpeed and eData.

Additional information on Topspeed's future direction is expected in Richard Chapman's talk on Wednesday, and Clarion Magazine will be providing more detailed reports and analysis in the coming days and weeks.

Vol 1, No 8
Sept 1999

DevCon '99 Special Report

TopSpeed Heads For Java

by Dave Harms

9/29/99 4:00 PM

Richard Chapman, TopSpeed's VP of R&D, announced this afternoon that TopSpeed will in a future release generate Java code. Although no firm dates or specifications were offered, the Java-capable version is the fourth product in the pipe, after Web Edition 2, the iBuild product, and C6, which will be a synthesis of the best of WE2 and iBuild.

Very little time was given to the Java product anouncement, which generally had a positive reception.

One of the biggest challenges facing TS in the move to the web is supporting distributed applications. By generating Java code, TS offloads language development and gains a lot of existing functionality which would be very difficult to reproduce in Clarion.

Chapman defined Clarion as TopSpeed's strategic language for client/server development, and Java as their strategic language for distributed, multi-tier development. One of these development strategies is in decline, the other is experiencing rapid growth. At some point the two lines will cross.

When asked if there is something happening between TopSpeed and Sun Microsystems (currently looking for a Java development environment), Roy Rafalco, Topspeed's President and CEO stated there was no such deal in the works.

Franks Watts also confirmed that although TopSpeed has several distinct products at the moment, the evolution is toward a single development environment. He also suggested that the optimal ugrade path is for developers to upgrade to the current Web Edition, then follow that path through to Web Edition 2 and then C6. WE2 will have a full copy of DreamWeaver integrated into the environment. This is expected to increase the cost by under $100.

TopSpeed also plans to open an online store using its forthcoming iSell technology.

Details on the Java strategy and the wrap-up sessions to follow.

BKO Enterprises, Inc.
Great Opportunity!
Salary to $125,000+
Pre-IPO Stock Opts
Sunny Boca Raton

etc 2000
If you're interested take our poll & let us know!