published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Volume 1 Number 10 - November 1999

Issue Index

[ABC Design Series: The ViewManager Part 1]
The ViewManager is the base class for all ABC browses. In this first of two articles David Bayliss explains ViewManager fundamentals and initialization code.
(Nov 23,1999)

[November 1999 News]
Clarion world news: product announcements, upcoming events, and more.
(Nov 23,1999)

[The Art Of Software Development: Analysis By Design]
Software development invariably means encountering and fixing bugs. But correction just leads to more correction. Forward progress comes from creating something new, which brings up the twin topics of design and analysis.
(Nov 23,1999)

[Clarion Magazine Version 2!]
Some of you may have noticed a few minor changes recently in the visual appearance of Clarion Magazine, particularly on the main page. Well, one thing led to another, and as a result in December you'll be seeing a brand new look on the web site.
(Nov 23,1999)

Vol 1, No 10
Nov 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

OLE The Easy Way: Part 3

by Jim Kane

Click here to read Part 1

Click here to read Part 2

When I was growing up my father often said to me he couldn't wait until I had a son of my own and he did to me some of the things that I did to my father. Well, my father got his wish and I have a son. Sometimes it seems my son's main goal in life is to help my father get even.

Not long ago I had the privilege of helping my son with a math assignment involving multiplication. While he was doing his work he kept longing to use a calculator and be done with it. As often as I explained the importance and virtue of doing it long hand so he learned the technique, the use of the calculator beckoned. As annoyed as I got at his insistence at using the calculator, I could not help but remember another little boy some 35 years earlier who thought memorizing multiplication tables was insane, and begged his father to let him use a calculator for math assignments.

Well, for much the same reason, in part one of this series I presented the assembler calling convention for calling OLE interfaces, and the Icall.A assembler procedure to call interfaces. Then in part two I outlined the needed API calls for calling interfaces. Now its time to "get out the calculator" and use the Icall.A assembler and some of the code and constants presented in part two to make a reusable OLE class for calling interfaces. While it isn't necessary to understand the material in parts one and two, it certainly would make your life much fuller and richer if you did!

To review from part two, the basic steps in calling an interface to create a Windows shortcut are:

1. Initialize COM (`CoInitialize` or `CoInitializeEX`)
2. Get interface pointers (`CoCreateInstance` and `QuerryInterface`)
3. Use interface pointers to create the shortcut (`IShellLink` methods)
4. Save the shortcut to disk (`IPersistFile` methods)
5. Release any interface pointers obtained (Release method of respective interfaces)
6. Uninitialize COM (`UnCoInitialize`)

Time to make a plan of attack for creating the class. As I read through the list, the data item that keeps popping up in the list is interface pointers. Managing these pointers and making sure they are released when I'm done is a constant headache with this kind of work. In addition, there are a few constants associated with each interface that are not much fun to track. Although not mentioned in the list specifically, in the code from part two there was a lot of work expended in detecting and reporting errors. Effectively managing the interface pointers would help with this.

Above and beyond the sexy interface pointers part of the operation are step 1 and 6. Gee, they certainly look like simple steps. Anyone thinking they are simple clearly forgot Microsoft is involved! In part two, I used

CoInitialize to initialize the windows OLE subsystem. Actually the preferred API according to Microsoft is CoIntializeEX(0,ThreadingModel).

The two most common threading models are Apartment and Multithreaded. In this case, IShellLink uses the apartment model. How do I know? By the apartment number in the phone book address? Not quite. Either threading model is known via documentation or the information is available in the registry.

To make matters worse, depending on the threading model the initialization goes in different places. For the Multithreaded model, initialization code can be done one time per application, so it's easiest to call it in the frame procedure. For the apartment model, the code needs to go in the same apartment or thread where the OLE code lives. For that model it's best to call the initialization code just before calling the OLE interface(s), and uninitialize just after.

Well, surely those two "simple" steps are done now. Guess again! There's one more wrinkle. CoInitializeEX() is not included in the TopSpeed lib files so you need to make your own library from Ole32.dll before CoInitializeEX() will link. One is included with the downloadable zip. With all this in mind, the class OLECl.CLW was written to carry out steps 1 and 6.

Figure 1. Code to carry out steps 1 and 6.

```
CoInit_ApartmentThreaded equate(2)
CoInit_MultiThreaded     equate(0)

OLEClType       Class,type,module('OLECl.CLW'),↵
  LINK('OLECl.CLW',_ABCLinkMode_),DLL(_ABCDllMode_)
!Member Data
fInitComm               long(0)
ThreadType              long(-1)
        ! 0=Apartment model; 2=Multithreaded; -1=illegalvalue
!Methods
InitOLE                 Procedure(long threadingModel),byte,proc
        !0 is good, else if fatal error
KILLOLE                 Procedure()
GetOleInitializedCount  Procedure(),byte
GetThreadingType        Procedure(),long
                End
```

The InitOLE method calls CoInitializeEX and increases fInitComm. The KillOLE method does the opposite: it calls UnCoInitialize and decreases fInitComm. On a good day, when your program is done, the count of calls to CoInitializeEX should exactly equal count of calls to UnCoIntialize and GetOleInitializedCount should return 0. That's on a good day. InitOLE should be called with a threadingModel parameter of 0 or 2 for apartment model or multithreaded respectively. Neither of the last two methods are needed in production code but may be valuable for debugging. Since it seems I spend most of my life debugging, that should make them very valuable indeed.

Well wait just a minute. Didn't I imply in the introduction that this was going to be easy? Lets take inventory. There's the Icall.A assembler module, the OLE32.Lib, OleCl.inc and Ole.Clw for initialization, plus if we ever get rolling here there will be one or so more classes to deal with those sexy interface pointers. Time to reach for Clarion's ace in the hole: templates. Also in the downloadable zip is OLETPL.TPL which adds all these bits and pieces to any application that will call an interface.

The template only has two prompts. First it asks for the threading model: Apartment or Multithreaded. In this case pick Apartment. There you have it, drop the OleTpl global extension template on an app and it's ready to call OLE interfaces. Oh, I did forget to mention one other detail: the second prompt asks for the name of the class that does the work of managing interface pointers. Looks like I'd better get busy building that

piece.

The main goal of the following class is to manage all the interface pointers, being sure to release all it acquires and manage any errors and the constants associated with the interfaces. All the code for managing the interface pointers and errors can be put into a general purpose helper OLE class. I decided to name it `holeClType` , short for helper OLE class. Nothing in that class is specific to this project; it's just general purpose interface pointer management.

The basis of the system is a series of equates, three queues and some beer:

Figure 2. Equates and queues.

```
!Interface equates
IShellLink                      Equate(20)
IPersistFile                    Equate(19)

!method equates with Method offset in the Vtable.
IShellLink_SetDescrip           Equate(IShellLink*100H + 1CH)
IShellLink_SetWorkDir           Equate(IShellLInk*100H + 24H)

VtableQType     Queue,type
VtableID          Long
pVtable           Long
                end

IIDQType        Queue,Type    ! pairs an interface with its IID
eInterface        long
IID_Address       long
InterfaceName     string(eDebugLabelLen)
                end

MethodNameQType queue,type
eMethod           long
MethodName        string(eDebugLabelLen)
                end

IBeerDrink      equate('JimKane')
```

One equate is created for each interface and for each method. In the code when I want to do any thing with an interface I use the symbolic name: `IShellLink` or `IPersistFile`. If I need to store or retrieve a pointer to pointer to the interface (see part one for an explanation) `VtableQ.VtableID` holds the equate and `VtableQ.pVtable` holds the pointer. The importance of this is seen in the release method code. When the job is complete and it is time to release all remaining pointers, all that needs to be done is loop through `VtableQ` and call the release method for each pointer that was acquired, like this:

Figure 3. The ReleaseAll method.

```
hOleClType.ReleaseAll        Procedure()
I    long
Recs long
    Code
    Recs = Records(SELF.VTableQ)
    Loop I = Recs to 1 by -1
       Get(SELF.VTableQ, I)
       If Errorcode() then break.
       SELF.Release(SELF.VTableQ.VtableID)
    end
```

Once all the interface pointers are stored in a queue, managing them is pretty easy.

Likewise in part two I had code that typically went like this:

Figure 4. Previous code to call QueryInterface.

```
!Get IPersistFile Interface by calling
! QueryInterface in IShellLink
hr = ICall2p(ppVtable_IShellLink, IShellLink_QueryInterface,|
   Address(IID_IPersistFile), Address(ppVtable_IPersistFile))
If hr<0 then
  Clear(ppVtable_IPersistFile)
  Res = Return:Fatal
  Do ProcedureReturn
end
```

One of the problems is if an error happens (and it always does if someone is with you watching), and do ProcedureReturn is called, you don't know which call failed. Ideally, if you could pair a string containing the interface name with the pointer to the interface you could make a more meaningful error message. The remaining two queues pair one of the equates for an interface or method with its name stored in a string. The IIDQ also stores the IID for the interface along with the string name. IIDs are needed in several of the API calls. The information is loaded by calling these methods:

Figure 5. Typical method calls for loading IIDs.

```
SELF.AddIID(IShellLink,Address(IID_IShellLink),'IShellLink')
! pair equate IShellLink with its IID
! and name stored as a string
SELF.AddMethodName(IShellLink_SetPath,'IShellLink_SetPath')
! pair IShellLink_SetPath equate
! for the IShellLink SetPath
! method with its name stored in a string
```

Now use the information stored in the management queues to redo the above call to QueryInterface:

Figure 6. The QueryInterface code.

```
hOLEClType.QueryInterface   Procedure(long eInterfaceToQuery, ↵
   Long pIIDNeeded, long eInterfaceNeeded )
pVtableNeeded  Long(0)
pVtableToQuery long(0)
hr long(0)
  code
  !using the interface equate, get the pointer
  pVtableToQuery = SELF.GetVtable(eInterfaceToQuery)
  If ~pVtableToQuery then Return Return:Fatal.   !abort on error
  HR = ICall2p(pVtableToQuery, eQueryInterface, pIIDNeeded,|
    Address(pVTableNeeded))
  If SELF.Failed(HR,'QueryInterface', eInterfaceToQuery, |
     eQueryInterface)     !make the call and handle errors
    Return(Return:Fatal)  !on failure return an error code
  else
    ! On success add the new interface pointer to the vTableQ
    Return(SELF.AddVtable(eInterfaceNeeded, pVtableNeeded))
  end
```

Notice the Failed method is called with the equates for the interface and method so it can use the information in the queues to manufacture a decent error message or progress message. Depending on a simple setting, Failed will display a message on both success and failure so you can follow the progress of the code, on error only for debugging, or display no messages for released code. For details examine the code for the Failed method in the downloadable code.

Notice in the above discussion of the three principle queues nothing was specific to the current project of creating a shortcut. For any specific project just take holeClType and derive the class to add the code specific to your problem. HoleClType also contains another helper class called strcl, short for string class, that helps with wide string to C or Clarion string conversions which you will also need. It also can handle bstrings but that code will not be needed for this project.

While I will not specifically discuss the string class one method needs special mention. When presented with a cstring that needs to be converted to a wide string, there is always a problem knowing how much larger a buffer or string to allocate at compile time for the wide string. To solve that problem the CtoWStrAlloc method dynamically allocates memory for the wide string output, and the caller is required to dispose of it. While not revolutionary, it is a valuable technique that may be of interest to some.

Very often when I write a class that I know will be derived each time it's used, I write a starter class that reminds me how to derive the class and of some things I may want to do. If the reader (that would be you) would like to follow along on how to build the final class, open the downloadable zip file and load OLEStart.Inc and OleStart.Clw. That is my "cheat sheet" for starting an OLE interface project. Just unzip the downloadable zip file to an new directory, register the template if it is not already registered then rename OleStart.clw and OleStart.inc to something related to your project.

In this case I used scut.clw and scut.inc. So the first task is to rename OleStart then do a search and replace for OLEStart and replace with ScutCl through both files. Notice Scut.Inc includes holeCl.inc and strcl.inc. As a result, I don't need to add those files to the project, just Scut.inc which in turn will add holecl.inc and strcl.inc. Notice also the Class(HoleClType) which tells the compiler this class is derived from HoleClType. With that done, Scut.Inc looks like this:

Figure 7. The ScutClType class.

```
!ABCIncludeFile

OMIT('_EndOfInclude_'_SCutPresent_)
_SCutPresent_ EQUATE(1)

!Other Classes
  Include('HoleCl.Inc')
  Include('StrCl.inc')

!Equates - will be seen by using program
ScutClType      Class(hOleClType),type,module('Scut.CLW'),↵
    LINK('SCut.CLW',_ABCLinkMode_),DLL(_ABCDllMode_)
Init                    Procedure(byte pDebugMode=0),byte,proc
Kill                    Procedure()
                end
 !_EndOfInclude_
```

As can be seen, there is nothing in this class other than the Init and Kill methods! The method I need is:

```
CreateShortCut Procedure(ShortCutStructType ShortCutStruct),byte
```

In support of that prototype I also need to add the ShortCutStructType that collects the information needed to create a shortcut:

Figure 8. Data structure for creating shortcuts.

```
ShortCutStructType Group,type
lpTarget              long     !can't be 0
lpDesc                long     !can be 0
hotkey                ushort   !can be 0 or a keycode
lpIconPath            long     !May not be null, but can use
                               ! same value as lptarget
IconIndex             ushort   !0 is the 1st icon
lpWorkingDir          long     !Can be 0 or NULL
lpLinkFileName        long     !file name with optional path for
                               ! the .lnk file to create
SpecialLocation       long     !special location to
                               ! create shortcut, 0=desktop
  end
```

With that added before the class definition, Scut.inc is ready!

Now, open Scut.clw and change OLEStart to ScutCl as before. Notice OleStart had places reserved to add the interface and method equates:

```
!Equates for Interfaces
!1-19 is reserved for hOleCl interfaces
    Itemize(20)
    !List all interfaces called here
    end
!Vtable Offset - List all methods called
!Method           equate(Interface*100H      + 0CH)
```

The method equate is manufactured from its interface name plus the offset down the vtable required to get to the method. This serves to make a unique equate for the method and stores the offset. To complete this task, fill in the list of interfaces in the itemize list and the list of methods in the method list. Once again, the method offsets come from .h files (Ask Bill syndrome again!).

The Init and Kill methods are fairly self explanatory. Since Scutcl is derived from hOleClType to be sure the hOleClType class's Init and Kill methods are called. The format is as follows:

Figure 9. The Init and Kill methods.

```
ScutClType.Init                Procedure(byte pDebugMode=0)
Res byte,auto
    code
    Clear(SELF)
    res = Parent.Init(pDebugMode)
    If ~Res then
       SELF.AddIID(IShellLink,Address(IID_IShellLink),|
          'IShellLink')
       SELF.AddMethodName(IShellLink_SetPath,|
          'IShellLink_SetPath')
      !Other interfaces and methods deleted for clarity
    end
    Return(Res)

ScutClType.Kill      Procedure()
  code
  Parent.Kill()
```

```
  Return
```

Notice the general pattern of:

```
code
  !code before parent call
  !Parent call - calls corresponding hOleCl method
  !Code after call to parent
  return
```

This ensures the parent holecl methods are called. As you may guess, on `Init` hOleClType needs to create the queues I described and destroy them on `Kill`. One of the side benefits of deriving a class is all these mundane details can be hidden in the base class or down in the 'hole' in this case! I just love to put the busy work out of site so I can concentrate on the rest.

After `CreateShortCut` validates the data, the next task is to get the two interfaces needed by calling `CoCreateInstance` and then `QueryInterface` method to get the second interface. Since calling `CoCreateInstance` and `QueryInterface` is something that is very common and happens many times in any OLE program, there are special methods for each in hOleClType. The code is as follows:

```
!Get the IShellLink Interface with CoCreate
 If SELF.CoCreate(Address(Clsid_IShellLink), IShellLink) |
   then return(Return:Fatal).

 !Get Ipersist_File from QueryInterface
 If SELF.QueryInterface(IShellLink, Address(IID_IPersistFile)|
   , IPersistFile ) then Return(Return:Fatal).
```

Take a minute and compare these two calls with the originals in part two. They're much less complicated. Progress! I may get out of this alive after all. The really nice thing is all the busy work not specific to this project is hidden down in `holeClType` and can be reused in any OLE project. In fact, I've used it in several projects already. Upon inspection `holeClType` has several "hidden" features not needed for this project but which are so commonly needed on larger similar projects that they are included.

The next section of code tells the `IShellLink` interface different properties needed for the short cut. This involves using the Icall.A assembler piece. However, you need not panic; it's well wrapped in `holeClType` so it should not be too scary.

```
!Set Short Cut Attributes
If SELF.ICall(1, IShellLink, IShellLink_SetPath, |
    SCS.lpTarget) Then Return(Return:Fatal).
If SELF.ICall(2, IShellLink, IShellLink_SetIconLoc,|
    SCS.lpIconPath, SCS.IconIndex) Then Return(Return:Fatal).
```

The first parameter (1 or 2) above tells Icall how many parameters will be passed to the interface. This is followed by the equate for the interface and method to call. In other words, the first line above calls IShellLink.SetPath(SCS.lpTarget) where SCS.lpTarget is the address of the file name to which the shortcut should point. This information is passed to the CreateShortcut method. All the details of getting the vtable pointers and method offset are taken care of in the holeClType never to be written again! On error, just return with a result code of return:fatal which signals the error condition.

The next section of code gets the path to the location where the shortcutshould go. This is often the desktop or some other special place. After calling two APIs to get the path, the code stores the path in the local variable szpath, a cstring. Unfortunately the last interface call requires a wide string. Since the length of the path string is unknown at compile time, the code will allocate heap memory for the wide string. The alternative would be to allocate a very large buffer certain to take care of any path. This is the ctowstralloc call in the string class mentioned earlier:

Figure 10. Convert to a wide string and save the shortcut.

```
!Now convert to a wide string
wszLinkFile &= SELF.StrCl.CtoWideStrAlloc(szPath)

!Last but not least, save the shortcut to disk
! then cleanup and exit:
Res = SELF.ICall(2,IPersistFile, IPersistFile_Save, |
  Address(wszLinkFile), True)
Dispose(wszLinkFile)
If res then Return Return:Fatal.
```

Notice after the final interface call that rather than testing immediately for an error, the dynamically allocated wide string is disposed of and after that a test for error on the last Icall is done.

As soon as the last call completes the shortcut is created. All that remains is to release all the interface pointers and clean up. Because all the interfaces are neatly stored in queues, to release them just loop through the queue and call the release method. This is already coded in the Kill method so all you have to do is call scut.Kill(). The code that actually goes in the application to create the shortcutis as follows:

Figure 11. Application code to create the shortcut.

```
OLECl.InitOLE(CoInit_ApartmentThreaded)
ScutCl.init(2)   !0=no errormessages, 1=msg on error only,
                 !2=verbose=msg on sucess and fail
Clear(ShortCutStruct)
!pgm to run when shortcut clicked
ShortCutStruct.lpTarget  = Address(Target)
!Optional descriptions
ShortCutStruct.lpDesc     = Address(Desc)
!Shortcut .lnk file name
ShortCutStruct.lpLinkFileName     = Address(LinkFileName)
!Optional working directory
ShortCutStruct.lpWorkingDir         = Address(WorkingDir)
!Take icon from target file
ShortCutStruct.lpIconPath         = ShortCutStruct.lpTarget
!0=Desktop
ShortCutStruct.SpecialLocation    = SpecialLocation
If ScutCl.CreateShortCut(ShortCutStruct)
   Message('Create Shortcut Failed')
else
   Message('Create Shortcut Worked')
```

```
end
SCutCl.Kill()
OLECl.killOLE()
```

When the demo SCUTABC.EXE is run the success (or failure) of each and every call is reported in a message since the ScutCl.init method was set for verbose. There are quite a few messages. To bad we don't get to bill customers by the step or message box! When ever you get tired of the messages just recompile with a debug setting in scut.init of 0 or 1. There are a few other goodies in `holeClType` for common OLE tasks such as memory allocation and expanded error handling which are available should you need them.

The nice thing is the next time out there is much less work now that the helper OLE Class is done. Just add the template, create the top level project specific class from OLEStart.CLW and call OLECL.INIT to initialize OLE. Then initialize the top level class and off you go.

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 10
Nov 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Clarion 5.5 Beta Sneak Peek

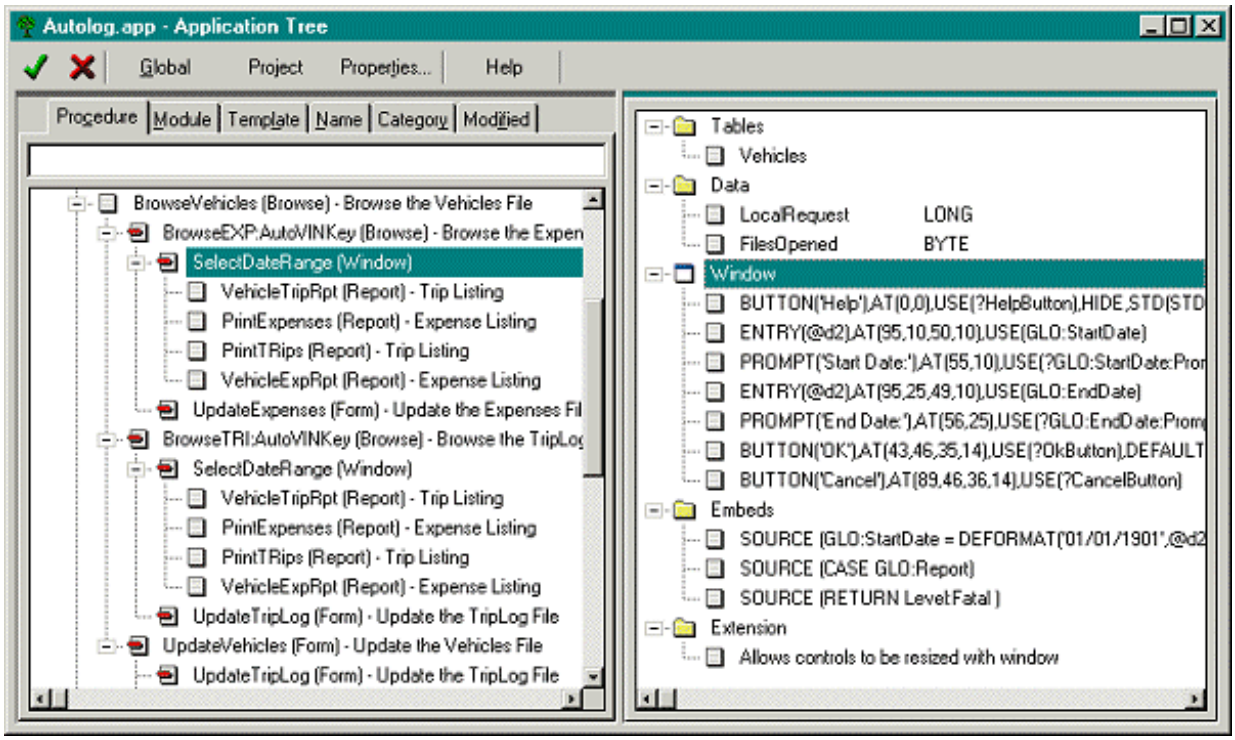
by Carl Barnes

As you've probably heard by now TopSpeed has released the first beta of Clarion 5.5. The main emphasis of this release is an all new implementation of a Java-free Clarion Internet Connect and numerous other internet features. But 5.5 also will contain many enhancements that will benefit every Clarion developer.

Application Tree

The first big change you'll notice in the AppGen is the Application Tree now has a pane on the right side. This is called the Split Pane View. It provides almost all of the procedure information without your having to open the procedure and dig around. The best part is when you double click on a row in the Split Pane View, the property dialog (or source editor) for that object opens. For example, if you double-click on an embed point, the editor opens with the embed source code. If you double-click on a window control, the control property window opens. Clicking on the Window folder opens the Window Formatter.

Figure 1: Application Tree (click here for a full-sized image)



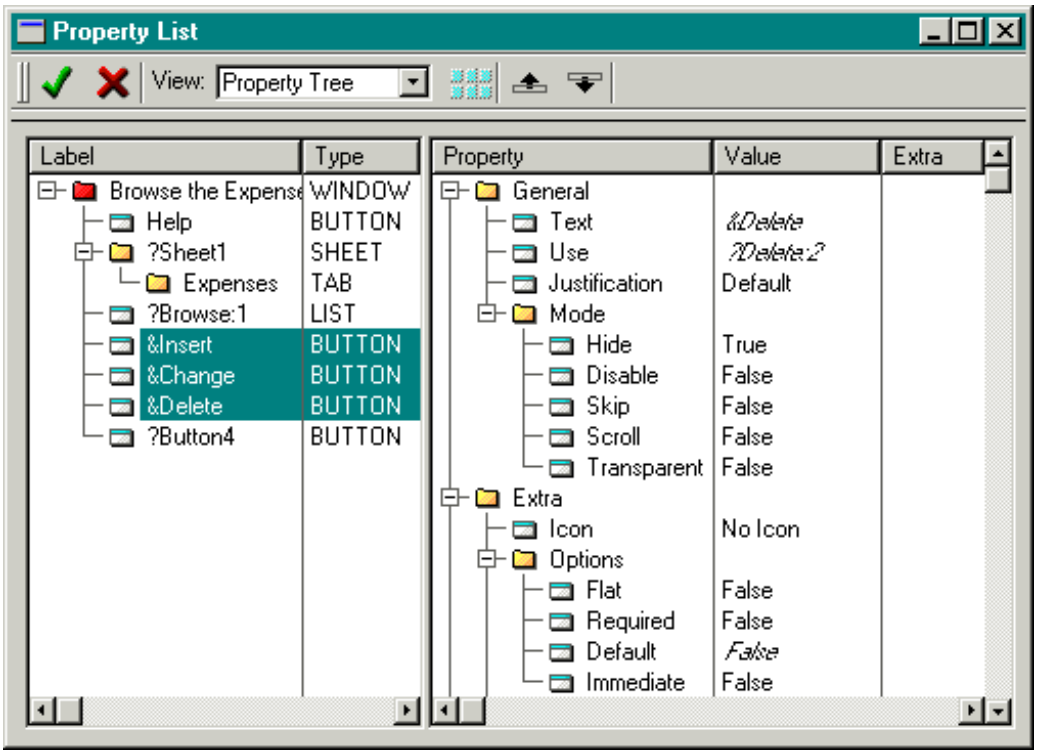
This looks to be a big productivity enhancement, especially for the many small changes made when testing and tweaking. You no longer have to drill down through multiple dialogs to get to the details. There are many possible enhancements that can be made to this new method of working. Only the mouse double click has been assigned so far leaving combinations with Control, Alt and Shift as well as the right mouse key for other shortcuts.

Also new is the Modified tab which lists the procedures in last modified date order. Often you will work on several procedures that do not sort together on any of the existing procedure tree tabs. This normally results in a lot of tab changing, scrolling and hunting around as you switch from one to the other. Recently modified procedures now appear near each other on this tab. And if something is broken it probably happened in one of the more recently modified procedures. This should be another timesaver. (If you are using VCS this tab also splits out Modifiable procedures from Readonly ones.)

Property Editors

New to the Window and Report Formatters are Property Editor windows that allow viewing and changing control properties in a list format rather than a dialog. These windows come in two flavors, the tree view shown in Figure 2 and the column view in Figure 3. The tree view is similar to the Split Pane View showing the controls on the left: when a control is selected, all of its properties are listed in the right pane. The properties are in a tree structured like the tabs and entries of the existing control property dialogs. This makes the transition to using the tree very easy. (The existing control property dialogs still exist.)

Figure 2: Window Property Editor Tree View

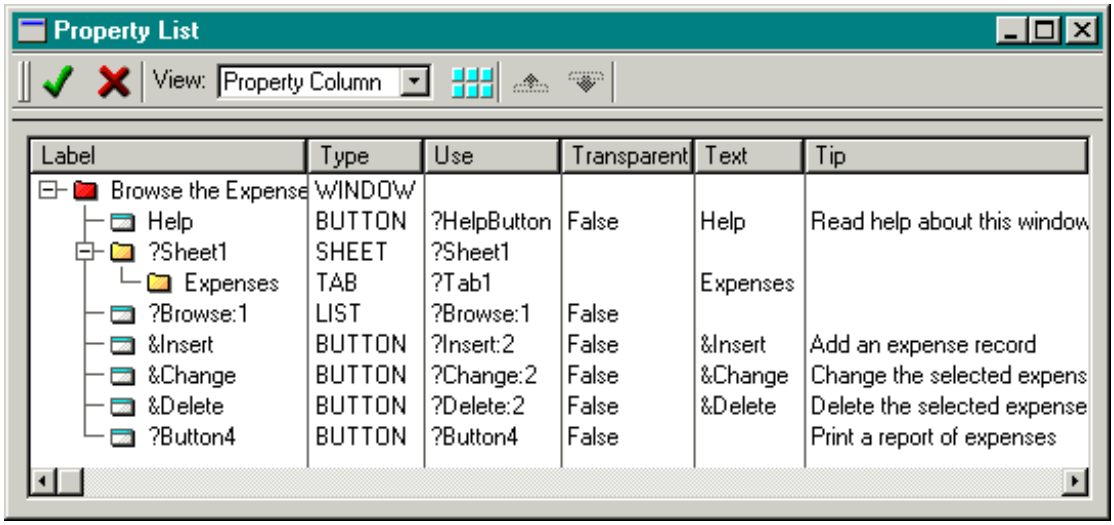


The Property Editor allows you to update multiple control settings at once using edit-in-place. In Figure 2 three buttons are selected in the left pane, and the properties they have in common are shown in the right pane. Properties that do not have the same value for all selected controls are shown in italics.

One other nice feature of the tree view of the window is that it shows you the exact structure of the window and order of the controls. On the toolbar are up and down icons that can be used to change the control order. This replaces the old Set Control Order... function on the edit menu.

The Property Editor's column view, shown in Figure 3, is like a spreadsheet with one control per row and its properties listed across in columns. You can define the columns to be displayed and their order. Also, the list of controls can be sorted by any column. This is handy for viewing or changing the same property across many controls when the value for each control could be different.

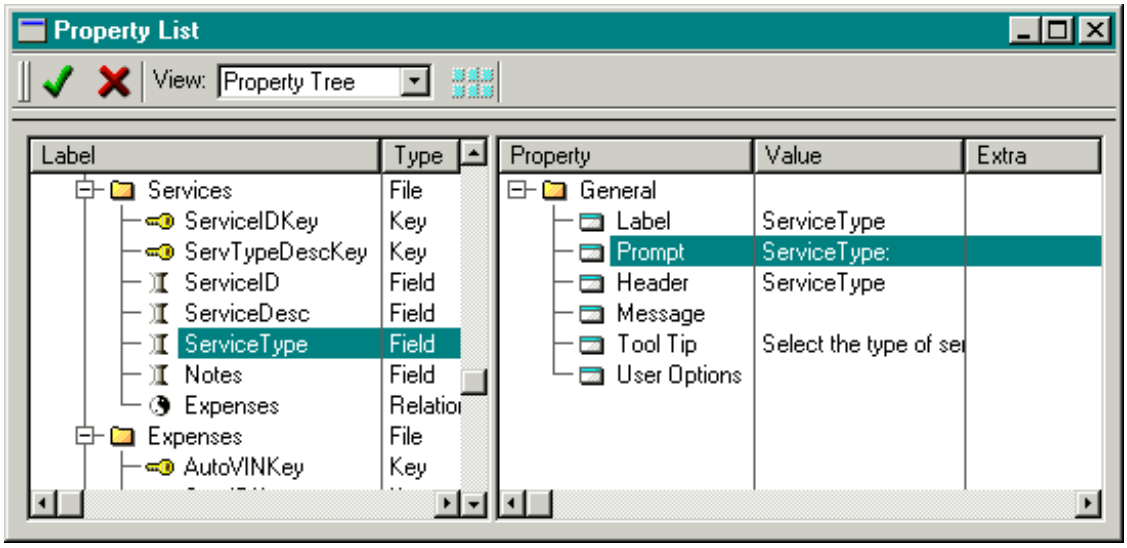
Figure 3: Property Editor Column View



In the alpha copy I tested the Column View did not yet allow editing the property values. Also the column picker needs work; in the alpha it has a simple list of field prompts in alphabetical order. For example, the prompt for the font name is "Name" which is not too descriptive and is shared by another field. I would prefer a tree structure.

The new Property Editor also shows up in the dictionary. It works in a similar fashion letting you view and edit file, field, key and relationship properties in one place. In the current alpha it is rather limited, only allowing the editing of the basic information shown in Figure 4.

Figure 4: Dictionary Property Editor Tree View

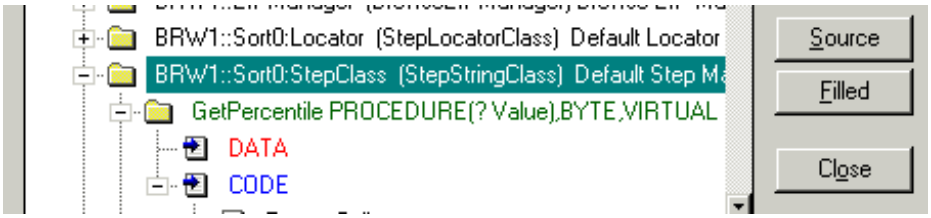


The tree on the left can be changed to a list sorted by label or type. There are useful options for putting all of your fields labeled "Name" together or sorting all of the keys together. The tree on the right also can be viewed as list sorted by property name or value. As in the Window Property Editor you can choose a column view and select multiple fields for making mass changes. Once this has more dictionary properties added it should be a very handy tool, letting you view and edit much of your dictionary in one single dialog.

Embeditor

New in the embeds list is a button below the Source button labeled Filled. This calls the embeditor the same way as the Source button does except only embeds that contain source are generated. This reduces the time to generate the source and cuts the clutter of having every possible embed point and virtual class generated. This is somewhat like the current module editor except you can edit the embed code. This will be another nice timesaver. It also will be a useful alternative to Module view when you want to read just the actual source of your procedure.

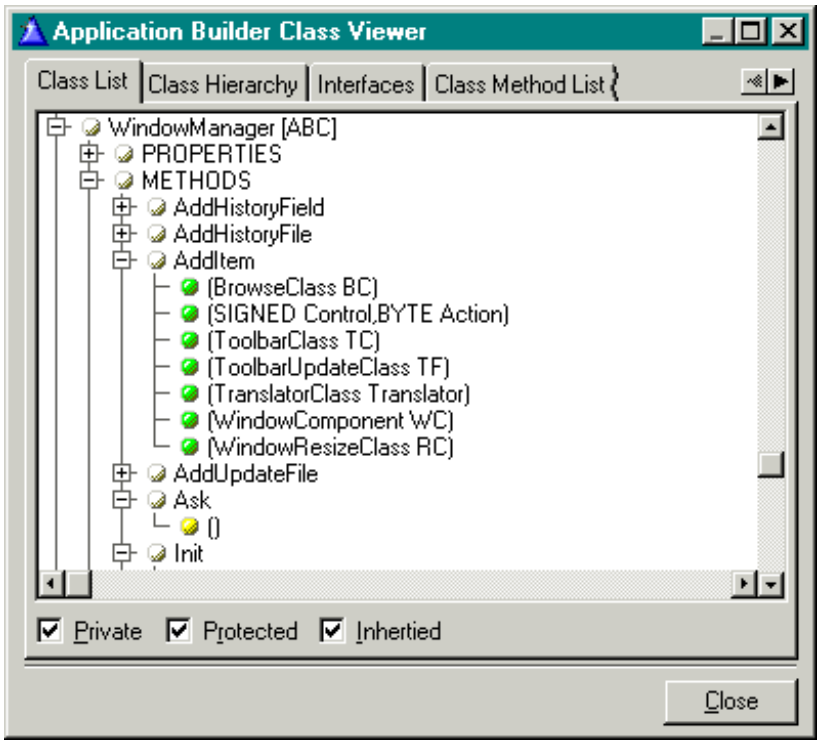
Figure 5: Embeds List



Class Viewer

The Class Viewer has been given a facelift and has a few enhancements. The colored text has been replaced by colored icons to indicate Private, Public, Virtual etc. (however there is no legend for the icons). There is a new option to omit inherited methods and properties. The files tab formerly showed just the classes contained in the file; now it shows properties and prototypes for each class. Interfaces are also shown on all views. More on that later.

Figure 6: Application Builder Class Viewer

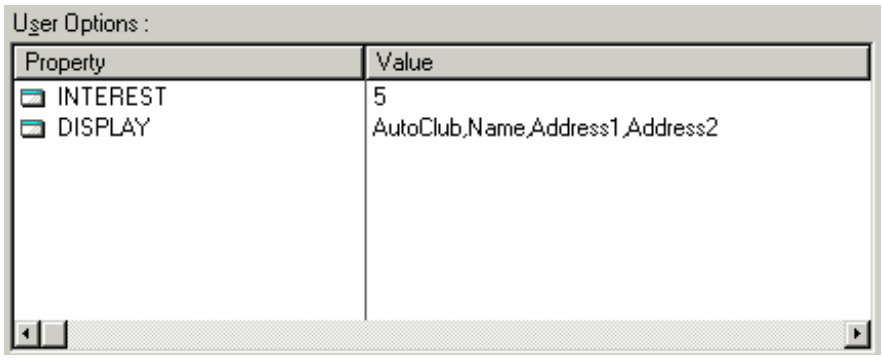


The interaction between the templates, the ABC classes and the AppGen have been formalized and improved. Every place in the IDE that requests a class name now has a drop list containing every ABC compliant class. In prior versions you had to type the name.

Other Dictionary Changes

The dictionary editor, which now has the aforementioned property editor, has also been given a face lift. The current text buttons have all been replaced by a tool bar at the top using icon buttons. User options, which are used extensively by the Wizatrons, are now displayed and edited in a table rather than a single text control.

Figure 7: Dictionary User Options



One other change to the complete IDE which is very apparent in the dictionary is the use of SQL terminology. "File" has become "table" and "field" has become "column." In the Split Pane View screen shot (figure 1) you'll see on the top right the Tables folder that lists the...well...it lists the files.

Interfaces

Clarion 5.5 implements the OOP concept of interfaces, which are also in Java (though Clarion's implementation is a bit different). An interface is just a special form of class declaration that has some special rules. An interface contains no properties (data) or code in the methods, just the prototypes for the methods. A class then `IMPLEMENTS()` the interface and provides the code behind the methods. *All* interface methods must be defined by the implementing class.

Listing 1 shows the interface between a window and a component of the window, for example a browse list. This interface allows the `WindowManager` class to control the component in a standard and specific way. In C5 the `WindowManager` has specific code for handling the browse class and every other class it controls. This has changed quite a bit in 5.5. Look at the `AddItem` methods and class reference queues it uses for each class it controls. For example, the window controls browses and file drops with very similar and redundant code and queues. Just the names change.

Code Listing 1: Interface Definition of Window Component

```
WindowComponent INTERFACE
Kill         PROCEDURE
Reset        PROCEDURE(BYTE Force)
ResetRequired PROCEDURE,BYTE      ! 1 if reset of window required
SetAlerts    PROCEDURE
TakeEvent    PROCEDURE,BYTE
Update       PROCEDURE            ! Everything but the window!
UpdateWindow PROCEDURE
  END

BrowseClass CLASS(ViewManager),IMPLEMENTS(WindowComponent)…

BrowseClass.WindowComponent.Reset        PROCEDURE(BYTE Force)
  CODE
  SELF.ResetSort(Force)

BrowseClass.WindowComponent.ResetRequired PROCEDURE
  CODE
  RETURN SELF.ApplyRange()

BrowseClass.WindowComponent.Update       PROCEDURE
  CODE
  SELF.UpdateViewRecord

ComponentList QUEUE,TYPE
WC &WindowComponent
          END

WindowManager.ResetBuffers PROCEDURE(BYTE Force)
I UNSIGNED,AUTO
  CODE
    LOOP I = 1 TO RECORDS(SELF.ComponentList)
      GET(SELF.ComponentList,I)
      SELF. ComponentList.WC.Reset(Force)
    END
!Above code is altered, replaced CL with ComponentList
```

Through the use of standard interfaces this code can be reduced to one `AddItem` method, one queue of `WindowComponent` interface references and one set of code for all classes that implement the interface. This eliminates a bunch of duplicate code and makes it possible to add new components to a `WindowManager` object without adding any new code.

In Listing 1 the 5.5 BrowseClass `IMPLEMENTS(WindowComponent)`. It now contains code for each of the seven methods of the interface (only three are shown). These methods simply call the appropriate Browse class method. FileDrops and any other components implement the same interface methods. When the Window Manager wants to reset component buffers it now uses the interface methods to do it. Previously the Window Manager called the Browse Class and File Drop Class `reset` methods directly.

Interfaces allow the Window Manager to be more abstract, contain less code and provide a standard framework for creating new classes that the Window Manager can control. This new additional to the Clarion language will allow a lot of expansion in ABC without the base classes growing into a tangled mess of redundant code. After reading and comparing the new ABC code many times it suddenly hit me how cool interfaces are. It's like DAB said, "understanding OOP takes about one second, after a few months of reading." If you don't get interfaces yet, keep reading.

Other Bells and Whistles

Here are some of the other new features in 5.5:

- Reports and process templates may now implement the QBE class.
- File loaded drop boxes inherit the `BrowseClass` to allow page loading
- The `FILEDIALOG()` directory picker in 32-bit is now the Windows standard control
- New `ASTRING` data type - see your Windows API under the topic of Atoms
- A procedure prototype can now specify that a string passed by address is a constant and cannot be changed by the procedure, e.g. `MyProc(CONST *CSTRING MyCString)`
- The `DATE()` function leap year bug has been fixed
- New enhanced version of VCS features a source compare tool
- New version 5.5 of Data Modeller
- Wizatrons have been refined extensively with many tweaks and improvements
- 5.5 will contain all the latest bug fixes and many minor enhancements

Product Registration: Late Breaking News!

The 5.5 Beta 1 has a copy protection scheme that requires registering each machine you install on and getting an unlock code specific to that machine. BUT WAIT! I have it from Roy Rafalco that this will be changed before final release. The new plan will require you to register your copy only once and you will get an unlock that will work on any machine. Should you have any trouble registering Beta 1 an unlimited unlock code is available from Team TopSpeed members or from TopSpeed.

This change to the registration process does not change TopSpeed's license agreement which requires you to buy one copy of Clarion per developer. This is an industry standard and very reasonable. You know it so don't play dumb. The price of Clarion is cheap compared to what it can produce. I spend much more on hardware and it does nothing for me without software.

If you are some company buying a single copy of Clarion (or my software!) for use by multiple developers I think you deserve a swift kick to the side of the knee joint. This should give you a life long limp to remind you how your short-sighted cheapness is ripping off software developers!

That wraps up the sneak preview of Clarion 5.5. The new Split Pane

View, Property Editors and Filled Embeditor features will shave time off your development efforts and provide you with better ways to view and mass-change application and dictionary data. ABC's better implementation in the IDE will help make the IDE a bit easier to use. The addition of standard interfaces to the Clarion language and their use in ABC will add more abstraction and reduce the code needed for new functionality.

You can see that London has been very busy adding features that make Clarion easier and faster to use. They have been spending much more time adding some incredible web features to 5.5. Read about it in next week's article by Steven Parker.

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

In This Issue

Vol 1, No 10
Nov 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Converting C5b Legacy Apps From TPS To MS-SQL 7.0

by: Stephen Mull, MCSE

This document relates my experiences and findings while performing a conversion of a large C5b Legacy app from TPS to Microsoft SQL Server. This document is based on Clarion 5b Enterprise Edition and MS-SQL Version 7.0. Using different versions of Clarion or MS-SQL Server may require different approaches, but information relating to file structures should still apply. I do not cover the details associated with MS-SQL 7.0 setup in this document, except where it relates to my Clarion application.

I have been using Clarion for Windows since the first release, and feel it is the finest RAD tool available anywhere! This was my first application involving the usage of a SQL based backend. While the initial learning curve has taken some time, I would certainly recommend the usage of SQL whenever possible.

My initial learning experiences began with reading the Clarion documentation, searching the newsgroups, and reading two really good articles relating to using Clarion with SQL. One is Rick Hoffmann's "MS-SQL Tips and Tricks and C5", and the other is a summary of the presentation by Scott Ferrett at Euro Devcon '99 in Amsterdam. While all of the available information was indeed helpful, there was no single source covering the details of my specific needs. I felt my experiences might be of some assistance to other developers, thus this article.

It is important to understand that you, the developer, must use your best judgment regarding the usage of this information. While I feel I am relating accurate information based on my experiences, your situation may be different, thus requiring different approaches. On we go!

MS-SQL Server 7.0, MS-SQL Server – Desktop Edition 7.0, MSDE, ODBC, MS-SQL Accelerator What Works And What To Use

What works and what to use? There are a variety of choices. After investigating the options available, I chose to use the Clarion MS-SQL Accelerator instead of ODBC. I also decided that MS-SQL Server 7.0 was the best choice. I would not recommend you use prior versions as Version 7.0 is far superior and free from headaches for the most part. The finished application will also work fine with MS-SQL Server 7.0 Desktop Edition.

Other approaches including using the MSDE or ODBC probably work fine, but my choices seemed to be the most reliable and straightforward approach for my project. Whatever choices you make, if you use Windows 95 as your platform you will need to install the DCOM updates first, and then the new MS-SQL ODBC driver (3.7x). The new ODBC driver is recommended with MS-SQL 7.0. These are all included with the MS-SQL 7.0 CD.

Changes To The Dictionary

The first thing to do is create a backup of *everything!* I suggest you start

with a new directory for the SQL app, and copy your existing app, dct, etc. to the new directory. You do not need to move the existing TPS tables into the new directory. You may use them later to copy the data to the SQL server, should you wish to do so. This will be discussed later in this article.

The first thing I had to do was make changes to the existing field definitions. Refer to the following table, which worked perfectly for me, excluding dates (to be discussed shortly).

The following table of field type equates is excerpted from Rick Hoffman's paper. Some changes and additions have been made to the original content.

SQL Field Type	Clarion Field Type
CHAR(20)	STRING(20)
VARCHAR(20)	CSTRING(21)
INT	LONG
BIT	BYTE
DATETIME	STRING(8) *OR* Date *OR* Time GROUP(Over String(8)) DATE ! you modify these fields TIME ! you modify these fields END
SMALLDATETIME	STRING(8) GROUP(Over String(8)) DATE ! you modify these fields TIME ! you modify these fields END
DECIMAL(18,4)	DECIMAL(18,4)
FLOAT	REAL
IMAGE	STRING(2048)
MONEY	DECIMAL(19,4)
NUMERIC	DECIMAL(18,4)
REAL	SREAL
SMALLINT	SHORT
SMALLMONEY	DECIMAL(10,4)
SYSNAME	CSTRING(31)
TEXT	STRING(2048)
TIMESTAMP	STRING(8)
TINYINT	BYTE

VARBINARY	STRING(255)
STRING	MEMO *see below

Global And Local Data Definitions

Don't forget to update all of your global and local definitions as well – they are easily overlooked! You will also need to add something along the lines of GLO:Owner for the owner name of each table (discussed below).

STRING To CSTRING

Look at the following string comparison: `'Dog' ~= 'Dog '`. With MS-SQL 7 tables, the trailing spaces become an issue when comparing field values. In most cases, this is only an issue of importance with key components as they are used with relational links and filters. I still recommend you change all of them accordingly. If you use CSTRINGs instead of STRINGs (highly recommended), then Clarion treats the trailing spaces in the same manner as the SQL 7 system. I encountered no problems doing this. Just remember that STRING(20) = CSTRING(21) in Clarion, so add the extra length in your field definitions or you may end up with truncated data!

MEMO To STRING

The MS-SQL driver does not support MEMO. Instead, you may create your memo fields as a very large string. As your SQL app is 32bit, you do not have a 64K record limit to be concerned with. After you make this change, you will find that you will no longer be able to display and update the "memo" contents in the form you used with MEMO. What you need to do to correct this is go to the DCT, go to that fields display properties tab and change the control type from Entry to Text. Then go to the form, repopulate the field onto the form, and it will work properly once again.

DATE - TIME To DATETIME

My DCT contains fields which store both date and time as Date and Time. I left these "as is", and upon sync with SQL 7.0 server, they were automatically converted to DATETIME and continued to function perfectly. Do not confuse DATETIME with DATETIMESTAMP on the SQL 7 server; they *are* different. I store my date values from within the app. This was appropriate for my application, but leaves the possibility of the data being incorrect if the workstation's date and time are not correct. Having SQL 7 insert date values is also possible, and will insure the correct date and time are used. If you do this, the SQL 7 Server will try to populate a DATETIME field with the date *and* time. It is possible to get around this, but I avoided it completely.

As I mentioned above, I store my dates and times separately. Clarion supports the DATETIME via a Group, as illustrated in the chart above, but reporting and filtering with it is a hassle I chose to avoid. Avoid storing various date and time data as a LONG. If you store as a DATETIME, your data will be usable with 3rd party products such as Crystal Reports, Access, Excel, etc.

Field Names And Definitions

I left my field names and key names the same, and all worked perfectly. There is a maximum field name length on MS-SQL tables, so be aware if you have excessively long field or key names. Regarding the use of external names, try to maintain identical field names. If for any reason the Clarion dictionary and the MS-SQL field names differ, you should set the External Name in the Clarion dictionary to the MS-SQL field name. Since you will sync from the DCT to the SQL 7 server, you should not encounter any issues in this area. Remember to update all field definitions to match the above list. One note about initial values and case: initial values will be set correctly using legacy code, even using recursive entries, but will not be set correctly using ABC, except on an initial insertAs far as case goes, it works fine, but keep in mind how this will relate to keys and NOCASE support, as mentioned below.

Of Keys And Indexes

Make sure all tables have a unique key. *This is very important!* Also realize that unlike Topspeed files, there is not a hidden record number in SQL 7 tables. You must inform the driver how to uniquely identify a record. To accomplish this, at least one unique key must be defined for each and every MS-SQL table. Do not use indexes with the MS-SQL driver, as they will not work properly.

Be sure to set all keys to either case sensitive or case insensitive. You must not attempt to use a mixture of both with your keys. The MS-SQL server's performance will suffer greatly if you do so. I learned this first hand! You must also not use GROUP fields as part of your keys. To restate this, fields within a group may be used as part of a key, but do not use any GROUP field in your key. I did not have any keys of this type, so I encountered no problems.

For files that required a unique record number or ID, I created a field called RecordID (@s18) for the file, and supplied its value from the app rather than the server. You may use the server to auto-populate this with an incremented number. To do this with MS-SQL, use an appropriate data type, most likely integer, and make it part of a unique key. Then mark the key as auto incrementing, exactly like you would in a Topspeed file. *NOTE: Topspeed and others claim this works, but I was never successful in getting this to work properly*. My method was to use Date() & Clock() on INSERT. Please don't call me crazy; it works fine with over 100 users adding records every moment of the day, there has *never* been a duplication error, and it requires no interaction from the server!

Referential Integrity

MS-SQL 7 does not have a cascade delete declarative referential integrity feature. I understand this is planned for the next version of MS-SQL Server. Thus, SQL 7 will not enforce RI except when you ADD a record. To use the server to update and delete child records, you will have to create triggers and/or stored procedures to handle the process. I found this to be quite a pain with SQL 7. With all of this given, I would recommend you do not change RI to Server based! I left the dictionary "as-is" here, and all works perfectly! Additionally, this allows easier data manipulation via 3rd party tools on the SQL 7 Server, but remember you can just as easily mess up your data using 3rd party tools to manipulate data, so be careful!

File Relationships

None of the documentation has properly addressed file relationships, in my opinion. I initially used the sync tool in Clarion, created a SQL script, executed on my new database on the SQL 7 server, and voila, everything was created properly including relationships. I encountered all types of erratic problems with my app with the relationships defined both in my DCT and on the SQL 7 Server. Even with newsgroup and Topspeed tech support, all of these issues could not be resolved. Others may argue my final solution, but it is working perfectly. What I did was to leave the relationships intact in my Clarion DCT, and *did not* create the relationships on the MS-SQL server. Everything works perfectly, combined with leaving the referential integrity as stated above. Additionally, this allows easier data manipulation via 3rd party tools on the SQL 7 Server, with the same caveat mentioned above.

Here is a helpful hint: *before* you sync you DCT with the SQL Server, save your DCT, then Save As a new DCT name, then Remove file relationships from the new DCT, save, then use the Synchronizer to create the SQL script (covered below). Be sure to use your original DCT with your app, and not this new one without the relationships!

MS-SQL Driver Properties and Settings

Here are a few tips I learned from Rick Hoffmann's paper and the newsgroups. See the Clarion docs for more information.

/SAVESTOREDPROC = FALSE

NOTE: sometimes the driver setting works backwards, so try

both. You will notice the performance difference! What you want to do is not have the server save the temporary stored procedures that your app will create on the SQL 7 server.

1. /TRUSTEDCONNECTION = your choice, based on your situation. I did not use NT security, instead opting for SQL Server security.
2. /LOGONSCREEN = your choice, based on your situation. See OWNER Attribute.
3. /GATHERATOPEN – Not used with MS-SQL Accelerator. This is for ODBC use only.

Driver Options

Change "Topspeed" to "MS-SQL Accelerator". Do this *after* all other changes are complete.

RECLAIM attribute

The MS-SQL Accelerator driver does not support this attribute.

Enable Field Binding Option

I enabled field binding in my DCT, and would recommend this to be enabled in most cases.

Enable File Creation Option

This does work properly with MS-SQL 7. I would still recommend the use of a SQL script instead.

OWNER Attribute

SQL 7 tables require an OWNER attribute. This indicates how to connect to the database where the tables are located. Use a variable whenever possible. If you have tables which already have an OWNER attribute, change them to something resembling !GLO:Owner. What do you do with this you might ask? See the Changes to the Application section below for information on how I handled this one!

Create a SQL Script Using The Synchronizer

To create a SQL Script you will need to first save your dictionary. Make sure you have completed all of the above actions before creating the SQL script, or you will have to do it again. You will probably not get it 100% right the first time, so don't worry! You will need your DCT file and an MS-SQL database. You get an MS-SQL database by either creating a new MS-SQL database or using an existing one. Make sure you have your MS-SQL server setup and running, and make sure you have installed the appropriate client software on your workstation. You may also use MS-SQL Server Desktop Edition if you do not have a separate server; it works fine.

I recommend you create a new database, using all defaults in Enterprise Manager. Also, save yourself some trouble and change the default sa database on the MS-SQL server login to your new database for the time being, or create a new login and give full rights to the new database to the new login.

Should you decide to follow my advice on not creating the file relationships on the SQL Server be sure to use a temporary DCT without the relationships before creating the tables, as mentioned above.

You run the Dictionary Synchronizer to create a SQL script, so first open your DCT, save, then run the Synchronizer. Select your other dictionary to be MS-SQL, and then select the database you want to create the SQL tables in. You will have to log in. Once you get to the Synchronizer screen you need to copy all the files to your SQL database. The easiest way to do this is to highlight the top line. Press the right mouse button and select add. Click OK, then click Finish when offered. You will be prompted for a script name and location, so answer accordingly. That's it! Run the script from Enterprise Manager's Query Analyzer, found on the tools menu of Enterprise Managers menu bar. Either load the script, or copy and paste from your script, select the database to run the script against in Query Manager and run it. You should be notified that the command(s) completed successfully in short time, and you're done! Now

migrate your data if you wish.

How To Migrate Existing Data

I decided *not* to create a conversion program for my data dictionary. I used MS-Access and the Topspeed ODBC driver, along with the MS-SQL ODBC driver from Microsoft to migrate all data, using append queries. It is quick and easy! Your existing data might require some massaging before appending to your new SQL 7 tables, so you could import the data into new Access tables, perform your data manipulation, then append the data to the SQL 7 tables.

Views

Please note my initial app did not utilize views, but they are easy to use in your app. Assuming you know how to create a view, do so on the SQL server, give it a name like v.myfile. In your dictionary go to the Synchronizer and import the view(s). Make sure you create a primary key, because when you import views a key is not created. Make the key look identical to the primary key of the view's main table. Make sure the following attributes of the key are set:

Require Unique Key = On

PrimaryKey = On

Case Sensitive = On

Exclude Empty Keys = Off

You may now use your view in your app!

PROP:SQL And Stored Procedures

Please note my conversion did not involve using any stored procedures. It is my understanding that working with stored procedures and other cool advanced functionality available from the MS-SQL server is best utilized via the CCS Client Server SQL Template Sets, available from Andy Stapleton at Cowboy Computing Solutions. I intend to purchase and add to my app very soon!

Team Topspeed recommends that when possible you should write your own PROP:SQL statements to process data. Please note I *did not* do this for my initial conversion, and all works perfectly, so I may decide to leave everything as-is. The Process templates *do work*! The MS-SQL driver creates stored procedures in the TEMP DB for all SELECTs and for all inserts, deletes and changes. PROP:SQL eliminates the stored procs for INSERTS, DELETES and UPDATES.

> *The following are some performance and usage hints for working with PROP:SQLs, excerpted from Rick Hoffman's paper). Some changes and additions have been made to the original content.*

Use performance hints such as NOLOCK, FASTFIRSTROW and INDEX = when using PROP:SQLs.

1. NOLOCK - Please read MS-SQL online documentation for description.
2. FASTFIRSTROW - Please read MSSQL online documentation for description.
3. INDEX - Sometimes MSSQL will not pick the correct Unique Constraint or primary key. Hinting will force MSSQL to use the specified PK or UC.
4. Don't use the RECORDS(TableName) on large tables to find the record count. Use a PROP:SQL with a SELECT Count(1) FROM TableName (NOLOCK). Since the select statement returns only one value you need to create a new file in the dictionary with a single field typed as a LONG.
5. For batch processes you may use BEGIN TRAN and COMMIT. Works like a charm, but monitor the transaction log for sizing (this is not a big issue with SQL 7).
6. NORESULTCALL - If the stored procedure is not going to return a result set then use the prefix NORESULTCALL. For example:

MyTable{PROP:SQL} = 'NOTRESULTCALL SP_UpdateWhatEver (1234)'.

7. CALL – If the stored procedure is going to return a result set via a SELECT statement then use CALL. Prefix is used. For example: MyTable{PROP:SQL} = Call('SP_SecuritySelectMembers (''FL0021002'')').
8. The data returned via a SELECT within the stored procedure needs to match the structure of MyTable. C5 only supports returning information via a SELECT. There are two ways to return things from a stored procedure. One is via an embedded SELECT and the other is RETURN (or similar command). The MS-SQL driver does not support returning information via RETURN, only SELECT.

Special Notes on PROP:SQL

The following special notes are excerpted from Rick Hoffman's paper. Some changes and additions have been made to the original content.

When issuing a PROP:SQL, issue a BUFFER(TableName, 0) before the PROP:SQL. This will tell the Cursor Fetch to retrieve one record at a time. Example:

```
BUFFER(TableName, 0)

TableName{PROP:SQL} = 'SELECT Field1, Field2, FROM TableName'

NEXT(TableName)
```

You can also issue a BUFFER(TableName, 20) and then Push the PROP:SQL twice. This will tell the Cursor Fetch to retrieve 20 records at a time. Example:

```
BUFFER(TableName, 20)

TableName{PROP:SQL} = 'SELECT Field1, Field2, FROM TableName'

TableName{PROP:SQL} = 'SELECT Field1, Field2, FROM TableName'

NEXT(TableName).
```

Issue a BUFFER(TableName, #) before the PROP:SQL.

When writing PROP:SQL statements its very easy to make syntactical errors. Team Topspeed recommended creating a debug procedure that's passed the SQL statement and displays it in a window. If you use a text field as the displayed field then you can cut and paste the SQL statement between your application and MS-SQL ISQLW or Query Analyzer. Example: SQLDebugWindow(TableName{PROP:SQL}).

Changes To The Application

In most cases very little change is required to an application for it to work with MS-SQL. Despite Topspeed claims that ABC is better than Legacy for SQL apps, I have found that Legacy offers excellent performance with MS-SQL 7.0. Additionally, due to MS-SQL's ability to self-tune, you will find performance will increase with each usage! I actually converted my app to ABC from Legacy, and have yet to eradicate all the minor bugs. I have decided to stay with Legacy for now.

As far as third party templates go, I use several third party template sets without issue. These templates include the following: CPCS Reporting Tools 5.1x, many templates from Sterling Data, some from Boxsoft Development, LSPack from Linder Software, Princen-IT Sendmail, and several other templates. I also use some custom templates that some very talented 3rd party developers have written specifically for me, and yes, they *all* work fine with MS-SQL! Be sure to read *all* Clarion docs referring to using MS-SQL Accelerator and the SQL accelerators in general. Also read any sections on Browses, Processes, Reports, etc. where SQL is referred. Although the docs are geared towards ABC, most information still applies in Legacy as well. You should actually do this *before* taking any of the steps described in this document.

Moving on to the app, one thing you will need to do is add a logon procedure at the beginning of the application. You will need to collect the SQL user id and password as a minimum. You *could* hard code this, hiding it from the user. If you decide to do this, remember to add some sort of message window telling the user to wait while the app connects to the server, as this could take a short bit of time.

Sample logon source is available in the Clarion docs, or you may create a Wizard app against any SQL database and the Wizards will automatically create the appropriate source code . Just cut and paste into your existing app. You may feel free to use the source in my sample app if you wish. An initial connection to the server is completed when any data file is opened, so open a file with your initial start. After opening any file, feel free to close it if you wish, as your connection to the server would remain until your app closes.

As mentioned earlier, MS-SQL 7 tables require an OWNER attribute. This indicates how to connect to the database where the tables are located. I created GLO:Owner as Global, CSTRING(255), and use an INI file to save and or retrieve and supply this string at runtime. I created a Clarion procedure that offers default SQL login definition, and store in the app's .ini file (this information could be stored in the registry for security or a TPS file with encryption if you wish). You must prompt your user to define the default login info using proper syntax. Refer to the Clarion docs for the Owner string format.

Embed source to retrieve the .ini entry and initialize GLO:Owner with this info in your main procedure before opening your SQL 7 tables. This way, if the string is empty, you will get the MS-SQL login. If it is populated correctly, the MS-SQL login will be skipped and login will occur automatically!

Another thing you will need to do is make sure that on any browse or process you set the Quick-Scan Records (Buffer Reads) option on. This will result in great performance increases! Any place you are prompted for an approximate record count, enter a number much greater than you actually ever expect to see with regards to records retrieved. If you leave this blank or zero, the process will first attempt to get a record count from the MS-SQL Server, which usually means a big performance hit!

Changing all browses to fixed thumb appears to enhance performance a bit, but records displayed in a browse sometimes disappear or appear twice. Your call on this one, as the book also said to do this. I did, but finally changed back. Also, make sure for any and all browses you have selected a key for the main file in the file schematic. This also applies to reports and processes.

If you find in a report or process that fields are coming up blank where you know data exists, check that the field(s) have the Bind attribute. This one tripped me up bad a few times! Be sure to check your embedded source or hand coded procedures if they involve data access, as these may require modification.

The browse and process templates access data through a Clarion view. This is why any field needed within a browse or process must have the "Bind" attribute. The easiest way to check things after making your changes is to run the app and see how it performs. Don't forget to address any auto-increment issues (mentioned above); these may throw you off as well! Also don't forget to bind any field used as part of a filter in your procedures.

Third Party Books And Reference For MS-SQL 7.0

Clarion Magazine published by Dave Harms. Fantastic Clarion resource!

SQL Server Magazine published by Duke Press. This too is a great new resource.

Using Microsoft SQL Server 7.0 by Que. This book is very complete.

Summation

These are the main things I had to address with my conversion. I did not have any other significant issues. I did have a bit of trial and error, and

you probably will too! Hopefully this article will help you avoid most of my trial and error process! Remember that I am not an SQL expert by any means! My methods were the result of reading everything I could find, asking lots of questions, and the trial and error process. If you are unsure about anything, post your questions to the newsgroups. Everyone is very helpful, and you will probably get the answers you need. I will make available for download a sample app, dct, and SQL script, along with a copy of this article in RTF format on my Website at: http://www.mullusa.com/demo/sqldemo.zip

Stephen Mull
http://ww.mullusa.com/products.html.

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Three Ways to Present
Many-To-Many Relationships
To The User

Part 1 of 2

by Tom Ruby

If your users don't understand your program, or don't like how you've presented their data, they're not going to be as productive as they could be (and they may not be your users much longer!) The more complex the data, the more careful you have to be about how you present it to the user.

One area that causes developers problems is many-to-many relationships. The concept of many-to-many has been previously covered in Clarion Magazine; in this article I'll cover three of the ways you can present this kind of data to your users. I call these ways "Form and Lookup," "Check List," and "Selection Pool."

Many-To-Many Basics

A many-to-many relationship describes a link between two entities, one on the left, the other on the right, where each item on the right may be matched to zero or more items on the left, while each item on the left may be associated with zero or more items on the right. Many-to-many relationships occur all the time. Some examples include:

- Which students are registered for which classes
- Which employees have which benefits options
- Which contractors have which skills
- Which facilities are reserved for which meeting

These are very easy to deal with using an intermediate *cross reference* table containing one record for each link with the record containing the primary keys values of both the other tables. This is easily demonstrated with a hokey example, and a very hokey example follows.

Penelope Puppylove has a dog act in the circus (didn't I warn you?). As part of her Dog Act Management System and Electronic Listing (DAMSEL), she wants to keep track of which puppies she has trained to do which tricks. Any puppy might be able to do several tricks, and she naturally trains more than one puppy to do each trick just in case one puppy is out of sorts for a show.

Immediately you can think of two tables, or files (I'll use the term table in this discussion, but for many flat-file database systems table and file mean the same thing). One table lists puppies, and one lists tricks. To deal with the many to many relationship between puppies and tricks you introduce a third table, often called a cross reference. I'll call it DoesTrick because it indicates which puppies do which tricks. Table 1 shows the three tables as I've set them up. You can look at them in the DAMSEL dictionary.

Presenting Many-To-Many Relationships
(Nov 9,1999)

Product Review: NiceTouch Dictionary/App Assistant
(Nov 9,1999)

Clarion 5.5 Web Development Features
(Nov 9,1999)

Table 1. Fields in the Puppy, DoesTrick and Trick tables.

Puppy	DoesTrick	Trick
PuppySysID LONG	DoesSysID LONG	TrickSysID LONG
PuppyName STRING(20)	PuppySysID LONG	TrickName STRING(20)
	TrickSysID LONG	

Notice that I've given each table a SysID field. This extra field is sometimes called a surrogate key, and its main purpose is to keep from having to put the PuppyName and TrickName in the DoesTrick table. Most experienced developers structure the data this way and make the SysID independent of any outside characteristic. If this was a database of people in the USA, you might be tempted to use the Social Security number as the primary key, but that isn't a good idea since social security numbers sometimes have to change and may be duplicated, often due to data entry errors.

I'll use the SysID as a primary key, and I'll set it as an autoincrementing key as well. The PRIMARY attribute is important for SQL databases, and the autoincrement setting causes Clarion code to be generated to do the autoincrementing.

The DoesTrick table could get away without a SysID since the two fields, PuppySysID and TrickSysID could make up a primary key. However I always put a separate field for each table's primary key because it is less trouble to have one you don't need, than to need one you don't have. SQL systems depend very heavily on the primary key to recognize which record is being updated.

Each table gets a primary key on its own SysID, and the DoesTrick table gets a key on the PuppySysID and TrickSysID fields. I also put a key on PuppyName and TrickName for convenience. The DoesTrick table gets two more keys, one on PuppySysID and TrickSysID, and the other on just TrickSysId. I marked the key on PuppySysID and TrickSysID to require a unique value so a trick can only be listed once for a puppy and vice versa. In some applications you might want to allow duplicates.

Two relations connect each Puppy record to many DoesTrick records by the PuppySysID keys, and each Trick to many DoesTrick records by the TrickSysID keys. The Clarion dictionary editor lets you define referential integrity to tell what you want done to the child records (DoesTrick) when the parent records (Puppy and Trick) are deleted or changed. Since I've used surrogate keys, there isn't any reason to change a SysID, so I have On Update set to No Action. I set On Delete to Cascade, so the DoesTrick records will disappear if the trick or puppy record is deleted. You could set the relationship between DoesTrick and Trick to Restrict on delete if you wanted to be sure a trick is never deleted if there is a puppy connect to it, which would be appropriate in many applications.

Notice that there are two ways of looking at the data. Penelope might want to look at all the tricks a puppy can do, or she might want to look at all the puppies that can do a trick.

The User Interface

Now that the data for this hokey application is defined, it's time to look at ways of presenting the data relationship to the user. The three options I use are "Form and Lookup," "Check List," and "Selection Pool."

Form And Lookup

In most Clarion programs the user edits a many-to-many relationship on the form where one of the tables is edited. A form and lookup interface is useful if there is extra information that needs to be stored in the cross reference record like how well the puppy does the trick.

NOTE: The example application contains completed procedures. If you want to follow along with the example application, you can either create new procedures with different names, or create a new example application and import demo app procedures as required.

If you create the DAMSEL data dictionary as I've described it so far, and let the wizards go at it, they will build you a Form and Lookup type interface. To show how such an interface works, I built it using the application generator. I started with a MDI frame as the main procedure with a menu option to call a browse of Puppies and another to call a browse of tricks, look closely at my DAMSEL.App. Each of these browses calls a form to update the puppy or the trick. In the sample DAMSEL application, I named these:

```
LookupPuppyBrowse
   PuppyLookupForm
LookupTrickBrowse
   TrickLookupForm
```

These probably aren't names you would use in a real application, but I used them here to mean "The example that uses a lookup to edit puppy," and "The example that uses a lookup to edit trick."

The main feature of `PuppyLookupForm`, in addition to the puppy name field, is a browse listing tricks the puppy can do. To build this browse, populate a browse box template onto the form. When the file schematic pops up, select the `DoesTrick` table, press the Edit button, and select the `PuppySysIDKey`. Since you want to show the puppy name and not anything out of the senseless `DoesTrick` table, hit `Insert` again, and select the `Trick` table from the list. Now, select the `TrickName` field from the `Trick` table. Now our browse box will show the names of the tricks rather than the `SysIDs` which Penelope doesn't know or care anything about.

Now go to the actions tab for the browse. Select the `PuppySysID` for the range limit field, set the Range Limit Type to File Relationship, and select `Puppy` for the related table. Just for grins, fill in `TRI:TrickName` under additional sort fields so the tricks will be listed alphabetically. (See Figures 1 and 2)

Figure 1. Actions tab for the trick browse on the puppy form.

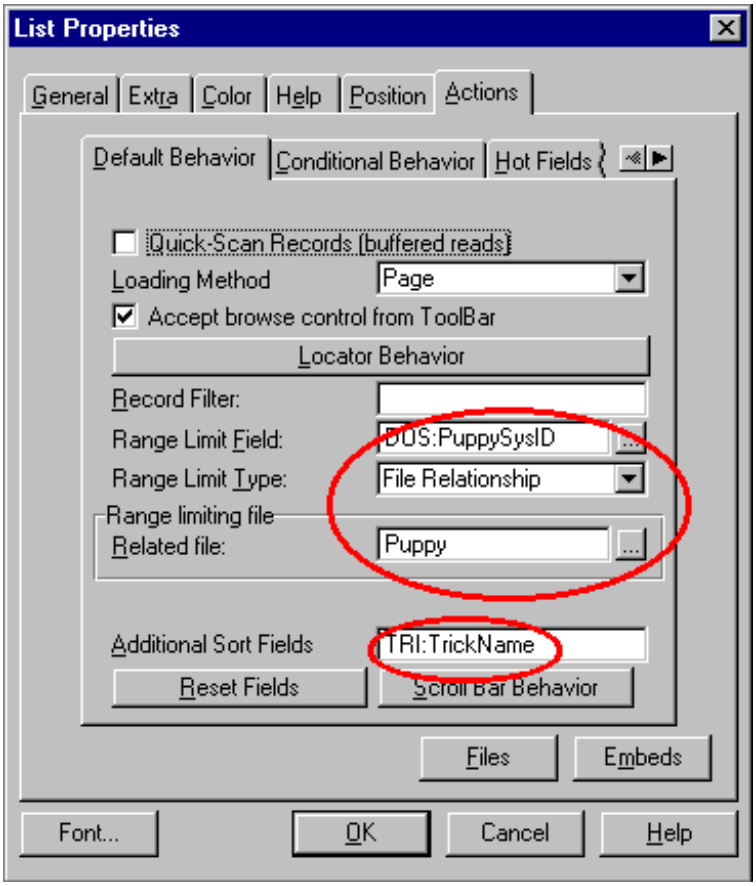
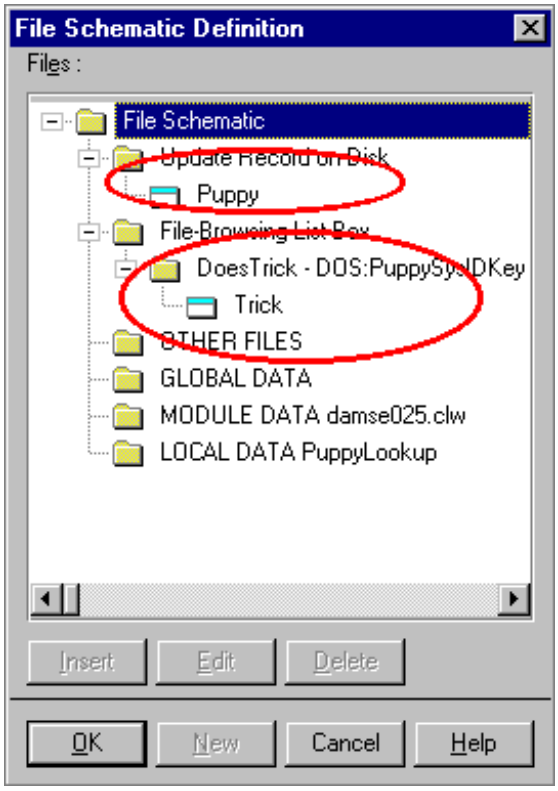


Figure 2. File schematic for the puppy lookup form.



Now the form will show the list of tricks the puppy can do. More precisely, it shows the trick names from trick table that match the `DoesTrick` records which contain the puppy's `SysID`. All that's missing is a way to edit the list. Populate a set of browse update

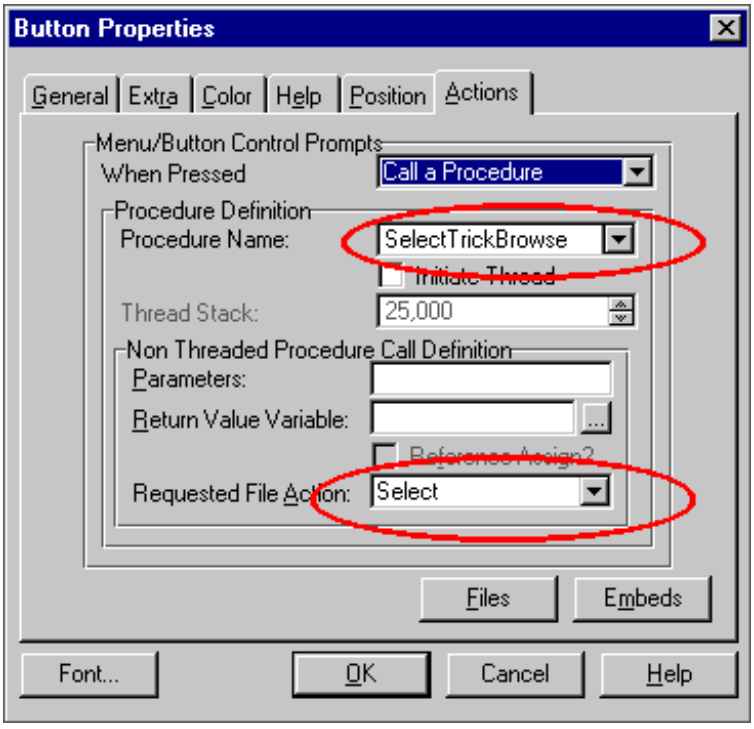
buttons, and set the update procedure to `SelectTrickLookup` (Select Trick for the Lookup Example). The working parts of this form are finished, so beautify it however you like and back yourself out to the procedure tree. I usually press the save button any time I get back out to the procedure tree, just for good luck.

There is now a `SelectTrickLookup` procedure labeled `ToDo`. Select this (or delete the existing procedure first to get a `ToDo`), and make it a form. This will be a very strange form since there won't be any entry fields. Instead, put a button, and label it Look up Trick. You'll have this button call up a list of tricks to select from, so on its action tab set When Pressed to Call a procedure. Fill in `SelectTrickBrowse`, for the procedure, and set the Requested File Action to Select. (See Figure 3) The `SelectTrickBrowse` will be a browse to select a trick. Now you need an embed to set the `DOS:TrickSysID` when the user has selected a record, so press the Embed button to get the tree. Open Control Events, Button3, and Accepted. Select Generated Code and press Insert to put the embed after the generated code, that is, after the call to the browse procedure. Fill in the embed to look like this:

```
DOS:TrickSysID = TRI:TrickSysID
```

You're done inside this form, so back out to the procedure tree, and make the new `SelectTrickBrowse` procedure a browse on tricks. Use the `TrickName` key so the tricks will be listed alphabetically. If you didn't use the browse procedure template, add the select button template.

Figure 3. Actions tab for the Look Up Trip button.



So how does Penelope use this thing? If she's looking at the sample DAMSEL application, she'll select Examples – Form & Lookup – Lookup Puppy. This gives her a list of her dogs with the usual Insert, Change and Delete buttons. Add a dog called, say, Phyadoux. Just hit the Insert button. The form pops up, and she puts the dog's name in. There are no tricks listed, so hit Insert to add one. The form with the Look Up Trick button pops up, and pressing the button shows an empty list of tricks. Only the Insert, Change and Delete buttons don't work. There's no trick editing form specified here and the program is assuming the tricks are entered somewhere else.

To make the program more useful, go back to `SelectTrickBrowse` and look at the Actions tab for the update buttons. Check "Edit In Place." This will let you add a new trick to the select list.

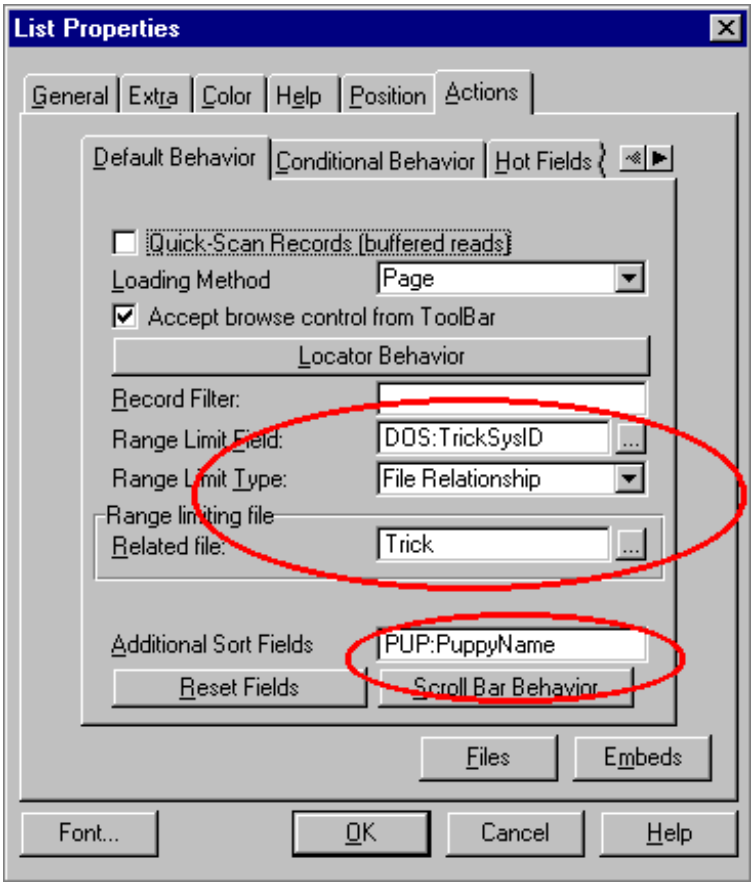
Now this application works, but to misquote Crocodile Dundee, "You can use it, but it works like..." Let's try making the Trick browse, form and lookup a little more graceful.

Select `LookupTrick` out of the procedure tree, and make it a browse procedure. In the File Schematic, select Trick for the file browsing list box, and use the Edit button to select the `TrickName` key. Then go to the actions tab of the update buttons and set the update procedure to `TrickLookup`.

Make `TrickLookup` a form, and select Trick in the file schematic under Update Record on Disk. Populate the trick name control. Then put a browse template to show the puppies that do this trick. In the file schematic for the File Browsing List Box, select `DoesTrick`, and use the Edit button to set the key to the `TrickSysIDKey`. Hit Insert and add the puppy file under `DoesTrick`. In the list box formatter, put the puppy name.

On the actions tab of the browse box, select the `TrickSysID` as the Range Limit Field, and set the Range Limit Type to File Relationship, and set the related file to Trick. (See Figure 4) In the actions tab of the browse update buttons, put `SelectPuppyLookup` for the update procedure.

Figure 4. Actions tab for the puppy browse on TrickLookup.



When you make the `SelectPuppyLookup` form, use a file loaded drop box instead of a lookup button and browse procedure. Ok, it stinks from a consistency aspect, but the idea here is to demonstrate different ways of dealing with these many-to-many relations. Make the `SelectPuppyLookup` a form, and select `DoesTrick` as the table being updated. Put a File Loaded Drop Box template on the form. The File Schematic will pop up, but select Local Data instead of selecting a file, Press the New button on the right side, and add a `LocalPuppyName` variable that looks just like the `PuppyName` out of the puppy table. I made it `STRING(20)` just for simplicity's sake. Select this new variable.

The file dialog will pop up again. Now it wants to know what to show

in the list, so select the Puppy table and use the Edit button to set the key to PuppyName. Then the list box formatter will appear. Choose PuppyName as display field and close the formatter.

The clever part of a file loaded drop box template is all on the Actions tab. You will find two important blanks on the Actions Tab, Field to fill from, and Target field. Set the Field to fill from to PUP:PuppySysID, and the target field to DOS:PuppySysID. When the user selects an entry from the drop list, the program will put the PuppySysID from the selected puppy into the PuppySysID of the DoesTricks record. When it displays, it will find the correct puppy by matching the PuppySysID from the DoesTrick record and show the name. It actually does this lookup from the list box queue.

When Penelope adds a puppy to a trick, this little form with a file drop box will show up, and she can select a puppy from the drop list. If I had used a drop combo instead of a drop list, using the same technique, I could have checked the Allow Updates box, and Penelope could insert a new puppy right there.

Keep three things in mind when implementing many-to-many relationships using the Form and Lookup technique:

- Browse the cross reference table, and show fields from the other related table.
- Range limit the browse to show only cross reference records matching the record being edited.
- Edit the cross reference table, selecting a record from the other related table.

Next week I'll show how to do this as a "Check List" and as a "Selection Pool."

Downoad the example application

Tom Ruby, who is no relation to the man who shot Lee Harvey Oswald, is an independent contractor living in the middle of a hayfield in Central Illinois with his wife Susan and two red headed sons, Caleb and Ethan. He has been using Clarion for Windows since the summer of '95. Before that, he was a "TopSpeeder" using Modula II, so he has never used the DOS versions of Clarion.

Presenting Many-To-Many Relationships
(Nov 9,1999)

Product Review: NiceTouch Dictionary/App Assistant
(Nov 9,1999)

Clarion 5.5 Web Development Features
(Nov 9,1999)

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Product Review:
Nice Touch Solutions Dictionary And Application Assistants, v1.01

by Tom Hebenstreit, Reviews Editor

I have decided that one of the truisms of programming is that every program starts off small and then proceeds to grow like a field of weeds as new features and requirements are added. For me, this unrestrained growth at the beginning is not really an issue, in that I can still keep all the necessary information about the application in my head as I work.

A major problem occurs once the app has reached a certain size, though; my brain does not share that same sort of infinite expandability (if only, if only...). The program becomes too large to comfortably wrap my head around – there are simply too many bits.

What do I do? Documentation, i.e., make lists. List of fields, list of procedures, lists of files: the list of lists is endless and becomes another problem in and of itself. <sigh>

So, if the key to maintaining control over my applications is keeping control of my lists, then it becomes very handy to have tools that can help to sort, report and categorize all of the major components of my apps. If they can help keep control of the myriad of other little details like tool tips, messages, descriptions, pictures and so forth, well, so much the better.

Now, you know this all has to be leading somewhere, right? And you are correct. The subject of this review is a pair of complementary products from Nice Touch Solutions, makers of justly famous Wizard line of tools (the Query, Report, View, CrossTab and Spreadsheet Wizards). Aptly named the Dictionary Assistant (DA) and the Application Assistant (AA), these two products work together as a team to help you document, understand and keep control of your applications and their dictionaries.

Note that although I will be referring to the pair as if they are basically one product, they can be purchased and used separately (there is a discount if you purchase them together). On we go...

Major Features

At their core, the Assistants simply expose all of the basic information about your dictionaries and applications in an easy to view and very flexible manner. You know the stuff I'm talking about – it's hiding behind a million property sheets, tabs and buttons all through the Clarion IDE.

To quote Nice Touch, Dictionary Assistant is:

"...a query and reporting tool designed for documentation, analysis and quality control of Clarion Dictionaries. Through the use of an application Utility Wizard DA creates normalized databases representing the components of the application dictionary (files, fields, keys and relationships). This utility wizard is a non-invasive Clarion Utility Template and therefore not part of your application."

Couldn't have said it better myself (so I didn't try).

Substitute "Application" for "Dictionary" and talk about procedures,

Presenting Many-To-Many Relationships
(Nov 9,1999)

Product Review: NiceTouch Dictionary/App Assistant
(Nov 9,1999)

Clarion 5.5 Web Development Features
(Nov 9,1999)

windows, reports, controls, templates and database file usage, and you have described the Application Assistant as well.

Both Assistants make heavy use of the Nice Touch Wizard products. Views can be modified and saved, reports can be created and saved, you can create and save queries on just about anything, and it is all done through a clean, consistent interface. (Nice looking, too.)

If you own TideStone Technologies (formerly Visual Components) Formula 1 ActiveX spreadsheet package, you get even more capabilities. Versions 4 and 5 of that product are fully integrated into both Assistants via the built-in Spreadsheet Wizard. Be aware, though, that if you don't own Formula 1 (F1), none of the spreadsheet related functions will work. (I'll be talking more about Formula 1 a bit further on.)

Implementation

My installs went smoothly, auto-detecting where Clarion 5 was located. After I had installed the Dictionary Assistant, the Application Assistant install automatically found that folder as well, offering to install in the same folder so that the two products could share the same runtime files. Very painless.

There's really nothing else to do other than register the appropriate templates. Once you have done that, you can open an application, run the utility templates and then get right down to business.

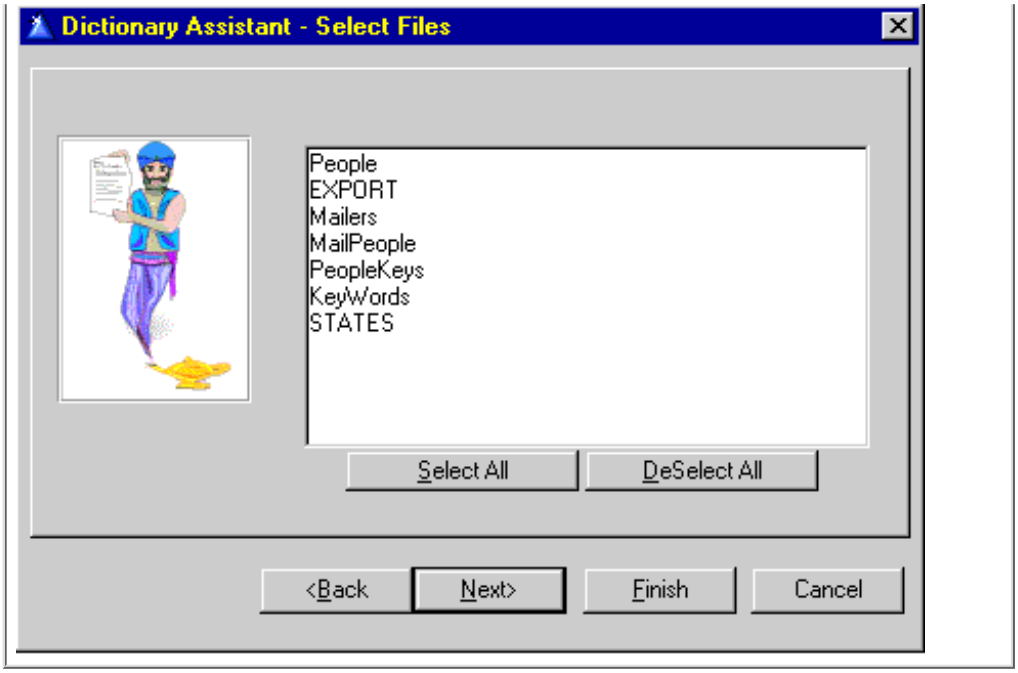
I did discover one issue when using the Utility templates related to long path names (short pathnames or names without spaces worked fine out of the box). It seems that the Assistants are not accounting for spaces in a long filename path when they are started using a command line parameter (which is the way the templates start them). Since they are using the Clarion Command() function to retrieve the filename from the command line, I found in the LRM that my long filenames/paths would need to have double quotes around them to keep them from being regarded as multiple parameters separated by spaces.

I edited two lines in each of the Utility templates to add double quotes around the file parameters, and after that everything worked fine no matter what the folder path was.

Performance

The process of running a utility template to export my dictionary or application information was fast and smooth. The templates fire up DA or AA at the end of the export wizard, so you are dropped right into the appropriate Assistant with the latest information.

Figure 1. Selecting files to export to the Dictionary Assistant.



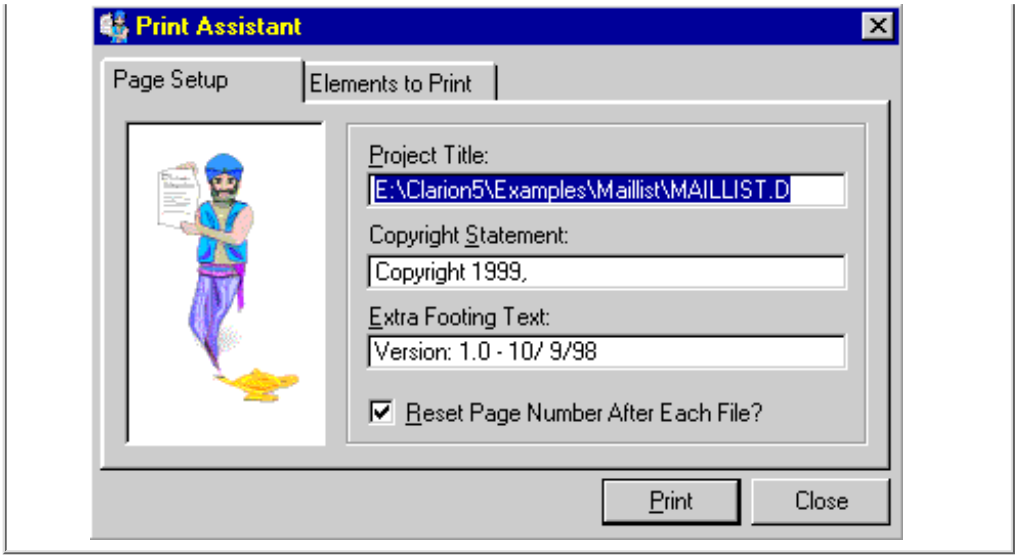
It is not necessary to select files individually, either. There is a simple checkbox to Analyze all files which I could have used, but I wanted to show a bit more of what the export wizard looks like.

From here on out I'll be referring quite a bit to the various Wizard tools used by Nice Touch to build these Assistants. As much as I'd love to show you lots of pictures, you'd probably hate me for the amount of time it would take to download them. On top of that, images of a wizard only show you one set of options (one tab) out of many. As a compromise, I highly recommend that you visit the Nice Touch Solutions web site, where you can download demos of most of the Wizard products at your leisure. Sorry – there are no demos of the Assistants, although you can download the help files (lots of pictures there).

Dictionary Assistant

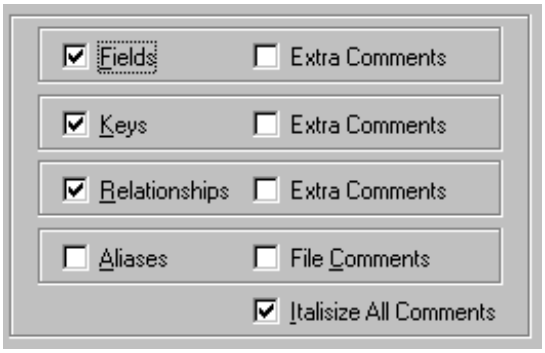
DA has two basic modes for viewing. The first, called Worksheet mode, has four list boxes that are tied together by a Files list. As you move around the list of files, the other three lists display all fields, keys and relationships for the selected file. The second mode is more free-form, and lets you browse lists of files, fields, keys and relationships in individual list boxes that offer an expanded range of Wizard options. You can run queries using Query Wizard, reformat the view using View Wizard, create custom reports with the Report Wizard and more. For quick and easy Dictionary printing, there is a very simple two-step Print Assistant as shown in Figure 2.

Figure 2. Panel 1 of the Dictionary Print Assistant.



Notice the option to reset the page numbers after each file. I really like this feature as it lets you keep a binder full of file definitions without having to worry about page numbering. Each file is sufficient unto itself, so to speak.

Figure 3. Panel 2 options of the Dictionary Print Assistant.



This figure shows the options from the 'Elements to Print' tab shown in Figure 2.

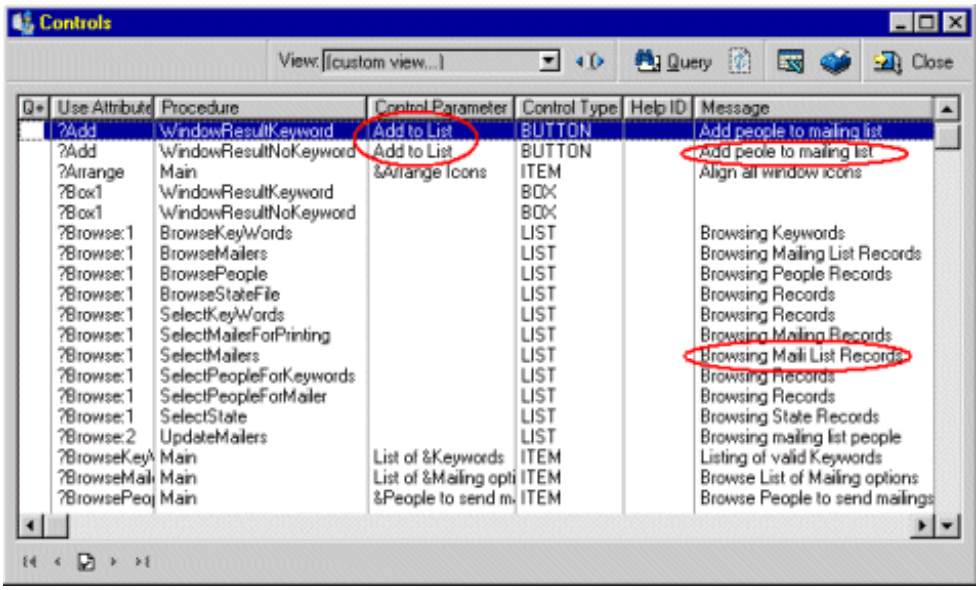
Pretty Printing...At Last

Printing deserves a bit of a special mention here. I've always had this theory that the person who wrote the built-in Clarion Dictionary print routines must own stock in a paper company, as they are not only rather awkward, but they are incredibly wasteful as well. Both DA and AA come with a large number of very nicely laid out and *useful* reports right out of the box. And since you can create, modify, copy and save your own custom reports, you can easily whip up just about any kind of listing you can think of. Add to that the ability to filter any report using the Query Wizard, and you will begin to see just how flexible these tools can be.

Application Assistant

AA functions almost identically to DA. It lists different types of information, of course, but the similarities make it very easy to pick up after you've used DA for little while. To illustrate just how useful a tool like this can be I'll illustrate with the stock C5 Maillist example. Figure 4 shows the Controls browse listing.

Figure 4. Application Assistant Controls listing (click here for a full sized image).



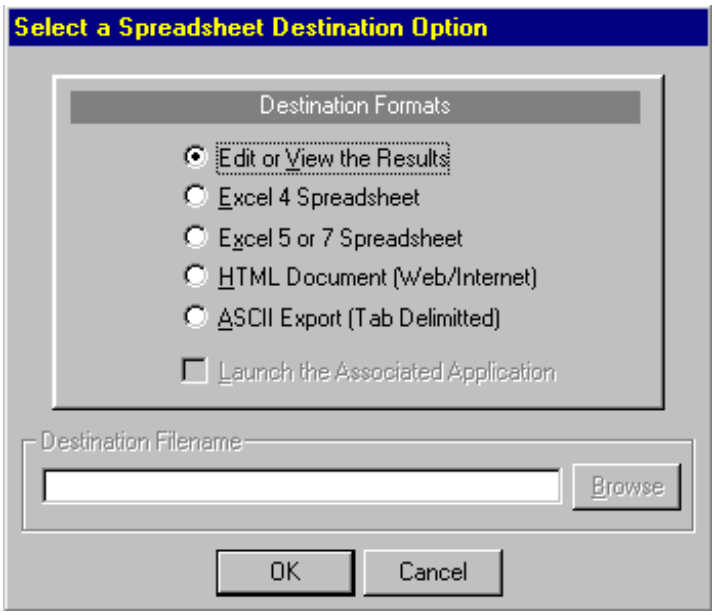
Note how easy it is to spot errors and omissions: there are two misspellings in the Messages column, and the Add buttons don't have accelerator keys. Imagine how long it would take you to track down those errors if you had to go procedure to procedure, window to window, control properties to control properties, then tab to tab. Yow! These are not errors that I created for the review either. You can go ahead and track them down yourself if you want to spend the time.

Also, be aware that you can sort on any column in every Assistant browse like the one in Figure 4 just by clicking on the column header. You can also drag and drop columns to rearrange the column order.

The Spreadsheet Wizard

Being the curious kind of person that I am, I just had to try out the spreadsheet-based options in the Assistants. After a bit of scrounging around, I found a demo of F1 version 5 and installed it. Sure enough, I could now view and manipulate my information directly in a spreadsheet. What was really neat about the Spreadsheet Wizard, though, was the ability to save my information directly into a number of popular formats (including HTML), as shown in Figure 5.

Figure 5. Spreadsheet Wizard output options.



But Wait! There's More...

Nice Touch also has a third Assistant, designed for viewing and printing Embed code. The Embed Assistant, though, is freeware, and can be downloaded at no charge from the Nice Touch web site. It doesn't have all the bells and whistles that the commercial products have, but I still found it to be quite handy.

Documentation

Documentation is supplied as Windows Help files, and I must say the files are very nicely done. Well organized, clearly written, and with an abundance of screen shots, they make it easy to learn about the programs.

I did find one rather glaring oversight, though: asking for help in the Application Assistant brings up the Dictionary Assistant help. You have to run the AA help directly from Explorer or via the Windows Start menu option created by the installer.

For the most part, though, the programs are very easy to use. If you are already familiar with how the Wizard line of products work, you'll feel right at home in no time at all.

Technical Support

Support is provided via the Web, Email and fax. In emailing Nice Touch regarding the few issues I came across, I invariably found the company to be both responsive and polite.

I'd have to say they live up to their name.

Room For Improvement

There are a number of little items that I found myself wanting while using the Assistants. Some are related to program use, such as wishing that the programs would remember what I was last working on, and open those files by default at startup. Another was to have the first column of some of the wider browses fixed on the left side, so that when I scroll horizontally there is no chance of losing my place (for example, the name of the field currently selected scrolls off the screen).

I also think that the programs should be a bit more informative as to what you can and cannot do. As they stand, they allow you to go through all the motions related to spreadsheets, right up to the point where you say "do it." At that time, though, nothing happens – no error (that's good) but also no message saying that you can't do this without Formula 1 (that's not so good). It would be nicer if they told you right off the bat that the function wasn't going to work.

About The Assistants And Formula 1 (F1)

Both of the Assistants incorporate the Nice Touch Spreadsheet Wizard, a tool that interfaces with the Formula 1 ActiveX Spreadsheet control from Tidestone Technologies. All is not roses in spreadsheet land, though. According to Nice Touch, Tidestone changed from a royalty-free pricing model for their older versions (4 and 5) to requiring user licenses for the newer version 6.x of F1. They (Nice Touch) have not yet fully integrated 6.x into their current products due to questions about licensing. Please note that no matter what the version is, F1 is *not* included with the Assistants–it must be purchased separately.

If you don't have one of the older versions, or can't find a trial version, you can download a 30-day trial of Formula 1 6.1 and use it with DA and AA (or your own Clarion programs, for that matter). Before you try, though, I'd recommend contacting Nice Touch and seeing what the current state of F1 support is. It may require some tweaking on your part to get 6.x to work.

How much does F1 cost? I found it for about US$86.00 on the web, and that also includes the bundled First Impressions charting control. That is for a version 6.x license, though, so you would need to buy

additional licenses if you want to use F1 inside your own programs (as opposed to just using it with DA and AA). The older, royalty-free versions (4 and 5) listed for between $250 and $300, if you can find them.

Summary

Each of the four components mentioned in this review is a standalone product, but as you begin to combine them, their usefulness increases exponentially.

The combination of the Dictionary and Application Assistants is really useful, and a worthy addition to any serious Clarion user's toolbox. Adding the Formula One spreadsheet raises both of them to another level, although it's a shame that there is the current uncertainty regarding supported versions. Toss the free Embed Assistant into the mix, and you've got a *very* powerful and flexible combination of Dictionary and Application reporting tools.

Nice job, Nice Touch!

PRODUCT RATING	
Overall	
Ability to do the task	Very Good
Ease of use	Very Good
Ease of installation	Good
Documentation	Good
Technical support	Very Good
Black box DLLs/LIBs	N/A

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

The Dictionary and Application Assistants can be purchased directly from Nice Touch via Telephone, Electronic Mail, Fax or Mail. Each Assistant lists for US$79, but you can also purchase them together as a bundle for US$129.

Sales Information, demos and more can be found at the Nice Touch Solutions web site: http://www.clariontools.com/Default.htm

For more information on the Formula 1 ActiveX control, visit: http://www.tidestone.com

Vendor Comments

Dictionary Assistant was originally developed to produce elegant dictionary documentation, suitable for delivery to large corporate and government contract clients. Over time we have decided to more fully expose the entire dictionary hierarchy. This "exposure" has aided our development efforts by providing a better foundation for beginning our application development (a strong dictionary).

We soon discovered an Application Assistant would provide a means for automating a significant portion of our quality control process. For example, the Exception Report within AA confirms the application represents the dictionary design. If we specified an @S30 in the dictionary and an entry form is using @S20, AA will tell us this before the client does! In a nutshell, AA's Exception Report produces a list of discrepancies between the Dictionary and Application.

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Clarion 5.5 Web Builder:
A First Look Under The Covers

by Steve Parker

"CWIC2," as it is called by some, is not the name of the Web Edition upgrade (aka Clarion5.5). It is not even the name of its IC component. Because it supplements and extends the existing Internet Connect technology, I think it appropriate to refer to the new extension as "Internet Connect, the Next Generation." But, I know better. If you check the template registry, you will see that it is listed as "Web Builder Templates." So that is how it shall be referred to here.

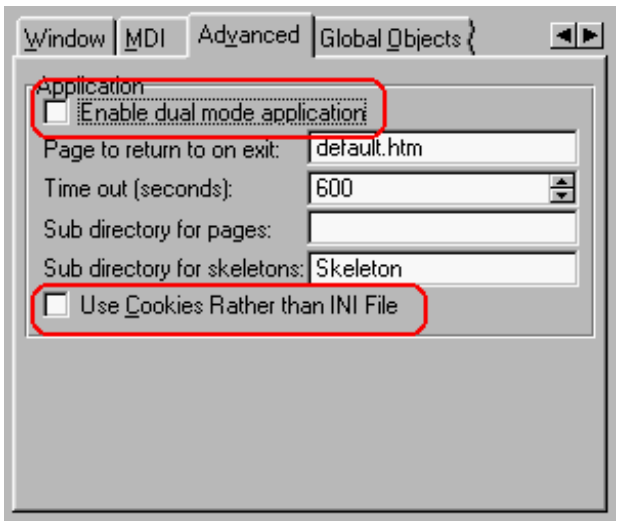
In every relevant respect the Web Builder templates are revolutionary for Clarion developers. Or, to be precise, the technology underlying them is.

Just how different is Web Builder from its predecessor?

For openers:

- Java is not even an option in a Web Builder app (bye bye .CABs and .ZIPs);
- Creating dual-mode apps is now an option (and not even the default option);
- The new INIManager is also a cookie monster; that is, on a PC or network, INIManager methods normally affect .INI files, but on the Web set and read cookies (cool).

Figure 1. Enabling a web application.



Different enough?

No? There's more:

- Web Builder is ABC only, having been fully absorbed into the ABC collective;
- To say that there are a myriad of new Internet methods would be the understatement of the week (see Figure 2);

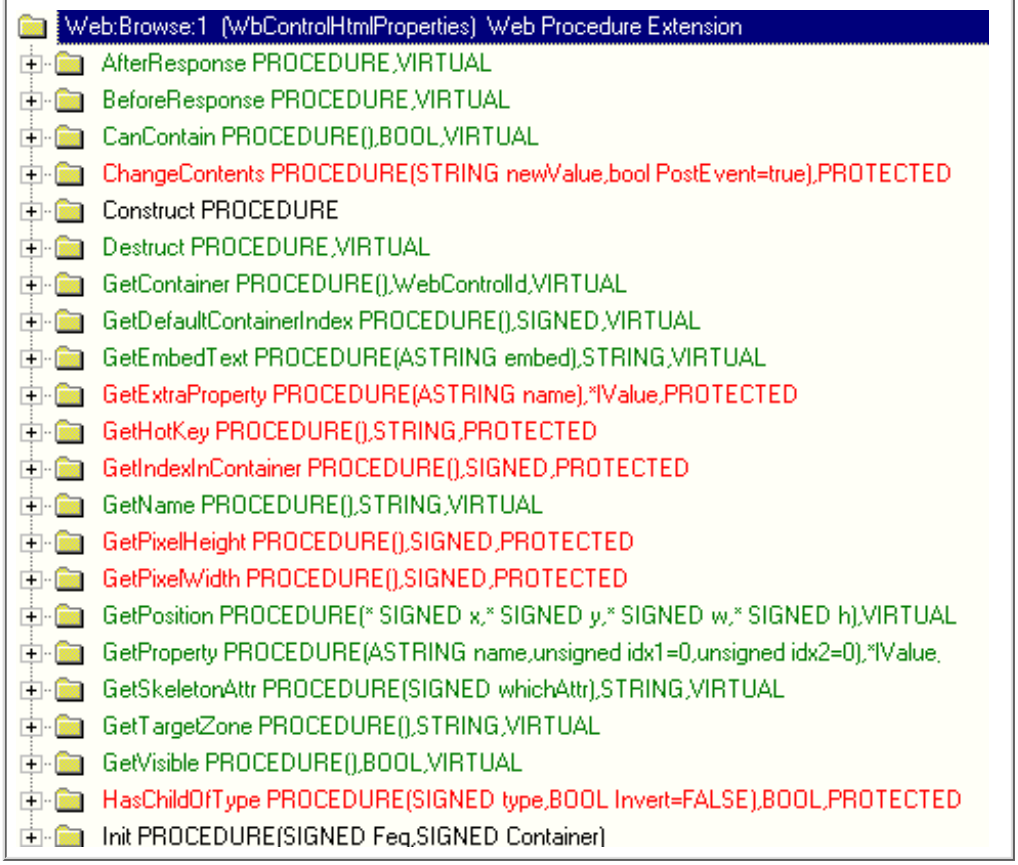
Presenting Many-To-Many Relationships
(Nov 9,1999)

Product Review: NiceTouch Dictionary/App Assistant
(Nov 9,1999)

Clarion 5.5 Web Development Features
(Nov 9,1999)

- If I decide to change the look and feel of an app, I no longer have to search through all the template prompts and embeds to find the code affecting the app's visual properties; changing one or two ASCII files will usually do the job (and they're not even part of the app).

Figure 2. New internet-related methods.



There has been a major philosophical change: Web Builder is designed so that a webmaster may change aesthetic aspects of an app, allowing full visual integration with an existing web site, without touching the actual executable or its functionalities. "Web-heads" can muck with an app but can't muck it up (at least, no more so than any other user). A web-head can make it look ugly but can't make it work ugly (only a developer can do *that*).

Different enough yet? Well, there's still more.

There is a built-in scripting language (TSSCRIPT) to help bridge Clarion to HTML. Scripts can be included without affecting the .APP or .EXE but only the generated HTML. That means that they can be added or modified independently of the .APP file.

Different enough?

Most visual tricks used with the previous version are no longer necessary.

However, if you need to be able to support every browser ever made, on every platform ever conceived (i.e., you are not allowed either Java *or* Javascript), you may not be able to use Web Builder. Web Builder, as delivered, relies heavily on Javascript (TSSCRIPT operates on the server side). So, if the "J" words are verboten, you may have to continue using the current Java-free techniques or, possibly, create a support structure that does not use Javascript. (The current Internet Connect templates will continue to ship with the product to cover this circumstance.)

In sum, Web Builder is an entirely new template chain with new method names and embeds. It is, simply, a whole new way of "going web."

Dual Mode

Before Web Builder, when you web-enabled an app, you automatically got a dual-mode application. With Web Builder, you must explicitly elect to create a dual-mode app. But if apps are not dual-mode, how do you test to make sure they work?

With the pre-5.5 IC product, you tested basic functionality in Windows. To test Web-enablement, you needed to have a web site, usually local to the development machine, available for testing purposes. You copied your app to the appropriate directory, started your browser and called your app. Then you tested again.

New in Web Builder is a "linked in broker." The linked in broker (LIB?) permits running an app locally on a PC without the full broker (apparently this is now called the Application Server, but you'll know what I mean if I continue with the older term, won't you?). The LIB does not eliminate the broker for Web deployment; it eliminates the need to create a local web site to test apps.

However, because LIB operates only locally (assuming, of course, that a browser and IP stack have been installed and that the appropriate DLLs and support files are available), an interesting consequence is that you *can* run an app in a browser locally. That is, you could deploy this way.

What happens is that, if your app is not dual-mode (i.e., is web only) and you press the make and run button, the broker will start, your browser will start and your app will be started in the browser.

Not only can you test functionality, you do so in a browser. This allows checking basic aesthetics and behavior in a production environment all at once.

Yes, if you double click on the .EXE from File Manager or Windows Explorer or from a shortcut on the desktop, the LIB will be called and the app started in a browser.

Convenient. *Very* convenient.

LIB can count: If you are already running a web server, the LIB will automatically bind itself to the next available port, port 81, before starting. If you are running a web server and have previously started the EXE broker, bound to an alternate port, the linked in broker will still bind itself to the next available port, port 82.

Very intelligent.

The New Paradigm

In Figure 1 you may have noticed a new template prompt: "Sub directory for skeletons." What is a skeleton and why does it need a subdirectory?

The new WB paradigm is actually a variation on something Clarion has been doing since Designer was first introduced in Clarion 2.0 (for DOS), something that has become so quintessentially Clarion that it often goes unremarked.

What is a Clarion .APP file? The standard answer is "a repository." And this is true. The .APP file is a repository of developer choices. It is also a database of those choices, options and entries and, in this guise, is used in a sort of mail merge. The Application Generator merges the templates with the data in the .APP to create Clarion source in the process we call "generation."

The new Web Builder paradigm is similar: it combines the .EXE with a set of HTML files located in this "Sub directory for skeletons."

Skeletons are simply HTML files accessible to (and deployed along with) your app. Each controls the appearance of a single web page element (corresponding to a window control).

There are skeletons for buttons, check boxes, entry fields, groups/group boxes, menu items, drop lists, menus, panels, prompts, radio buttons, regions, sheets, strings, tabs, text fields and

browses. There is also one providing the general structure for a window, another containing common functionalities for all windows and one for colors of different page areas.

At runtime, these HTML files are read and their properties stored. When pages are actually created, window elements are merged with the appropriate HTML skeleton based on the properties best matching the control's requirements. This "merging" produces the final page element.

So, a skeleton is, in fact, a sort of control template for producing HTML.

> *To take advantage of the linked in broker, your skeletons must be visible to your app. Typically, this means they are deployed in a directory below the development directory (you simply name a directory in the template prompt – you may hard code a reference to another directory, should you wish). The necessary files are installed to a subdirectory called \DISTRIB. A PUBLIC directory will also be created and, as before, images are deployed there.*

In short, the app (still) controls behavior and basic control location; skeletons control and fine tune appearance.

The flexibility is that the appearance of an element can be modified by changing, not the app, but the relevant skeleton.

For example, DisplayText.htm

```
<!-- DisplayText.htm -- Start --> <TSSCRIPT
value=DisplayText> This is the text </TSSCRIPT> <!--
DisplayText.htm -- End -->
```

controls the production of prompts and strings. If you would rather have those controls display using Verdana bold, just change DisplayText.htm:

```
<!-- DisplayText.htm -- Start --> <font size="2"
face="verdana"><b> <TSSCRIPT value=DisplayText> This is
the text </TSSCRIPT> </b></font> <!-- DisplayText.htm --
End -->
```

and, voila, prompts and strings now use that font.

In light of the prompt in Figure 1, it is should be possible to have multiple skeletons affecting the same page element. Multiple skeletons would be used to make different procedures look different from one another. It should also be possible to use different skeletons for different controls of the same type within a single procedure or within a single app. That is, it should be possible to nominate alternate skeletons in two different ways.

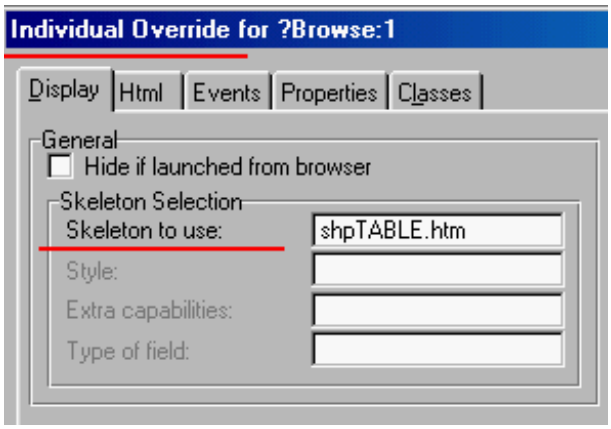
And, indeed, this is the case.

Obviously, you may simply modify the shipping skeleton. In this case, all controls constructed from that skeleton will exhibit the properties specified in the modified file. Using the modified DisplayText.htm, above, all prompts and strings in all apps executed against the same skeleton directory would be Verdana bold.

If, for example, you prefer prompts and strings in Arial, modify DisplayText.htm accordingly and it will be made so the next time you access the app in a browser.

Additionally, you may create a new skeleton, based on an existing one and save it to a different name (or, same name, different directory). Then, on a control-by-control basis, name the skeleton to use:

Figure 3. Specifying a skeleton to use.



In this case, only the one control will be different.

Now, that's power. And skeleton mods can be done on the fly too (at least as long as no app is actually accessing the skeleton at the moment you save it). No matter how you cut it, this is vastly easier than a re-make. (Besides, the broker lets you stop an app and web-heads keep even weirder hours than we do. So, teach them how....)

Oh! Did I mention that skeletons can be customized in a visual HTML editor?

Summary

It sounds simple: the Clarion app controls functionality and placement of page elements, the skeletons (official name, "WebStyles") control appearance. This appears to delegate the substance to the Clarion developer and the superficialities to the web-head.

As a long-time subscriber to the Theory of Competitive Advantage, this strikes me as entirely appropriate and just.

Unfortunately, it doesn't always seem quite that simple. The very simplicity of the interaction easily confuses many.

In coming articles, I'll peek a bit deeper under the covers and expose more of this interaction.

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitor's right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Presenting Many-To-Many Relationships
(Nov 9,1999)

Product Review: NiceTouch Dictionary/App Assistant
(Nov 9,1999)

Clarion 5.5 Web Development Features
(Nov 9,1999)

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Relation Trees: A Few Of The Finer Points

by Steve Parker

"Write about what you know." That's the advice Randy Goodhew gave me...more than once. Often, however, I didn't listen. Instead, I wrote about my learning curve as I worked through something new.

So it is with relation trees.

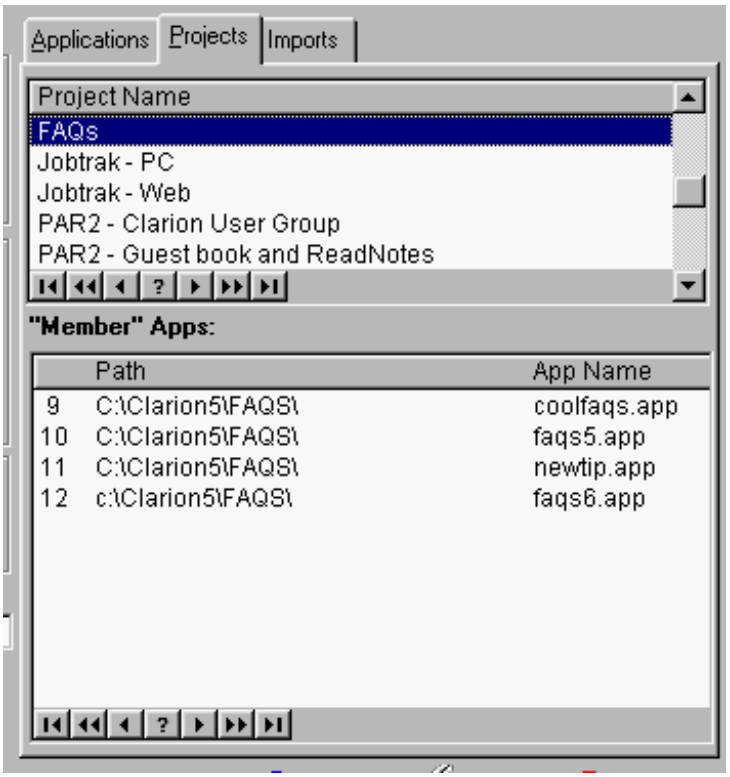
Relation trees are "not appropriate for databases with a very large primary file" (*Application Handbook*, p. 136). I once tested a relation tree on a 2000 record file. Performance was unacceptable. I have not looked at them since. Until recently, that is.

Why Now?

What suddenly got me interested in relation trees? The short answer is "wrist pain".

My batch compiler, with which I automate making multi-DLL applications, has a tab for "Projects." On this tab, I have a standard parent-child browse pair. In the top browse are the projects and in the bottom one, the applications comprising the project (Figure 1).

Figure 1. The pre-relation tree batch compiler



Relation Trees: A Few Of The Finer Points (Nov 16,1999)

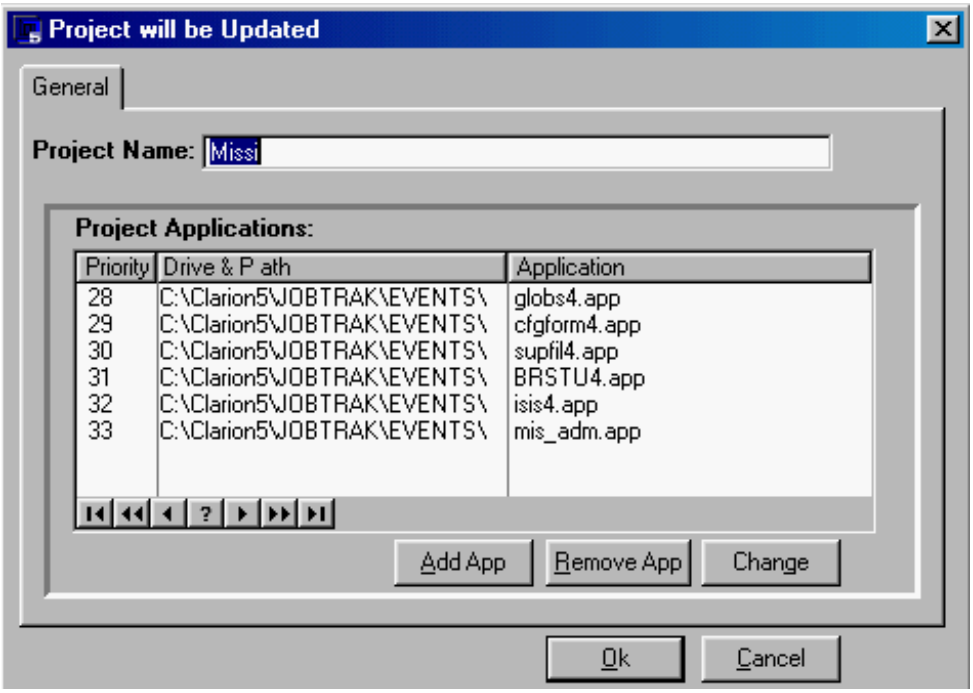
Presenting Many-To-Many Relationships: Part 2 (Nov 16,1999)

The Clarion Advisor: Breaking Out Of Nested Loops (Nov 16,1999)

As you will notice, the project browse can only show five rows before scrolling becomes necessary. Of course, I recently found myself changing, repeatedly, the dictionary for a project towards the bottom of the list. Of course this requires a complete re-compile of all of that project's component apps.

Since even the most prolific and in-demand Clarion programmer will not have more than 15 or 20 projects (sets of applications, primarily involving multiple DLLs, as opposed to having only 15-20 applications), the maximum number of records in the Projects file will almost always be reasonable. A relation tree could (probably) handle this many records. And, if my Project components list were in a tree, I could use the entire tab for a single Projects browse and minimize my scrolling requirements, making life easier on my mouse wrist. And because a relation tree can be configured to allow updates at both parent and child levels, I could continue to maintain my Projects file from the tree using the existing update form (which can do both Project and components in one go – see Figure 2).

Figure 2. The existing update form.



What I had in mind would look something like:

Figure 3. The compile manager with a relation tree (list box only - click here for the full image)

(And, in fact, if you download this program, this is exactly what you will find. Consider this the official announcement of the enhancement.)

So, come along as I learn some of the subtleties of using the relation tree template.

Tree Basics

Everyone has seen tree controls. They have plus and minus signs; they can have icons. They can show multiple levels of a single file chain. They can be expanded and contracted (usually at user request but sometimes magically, unbidden). But the single most important fact about trees, as implemented in Clarion, is that "The Relation Tree template employs a fully-loaded QUEUE for the root level" (*Application Handbook*, p. 136).

Relation trees are QUEUE based.

This means that everything I know about Clarion QUEUE structures can be brought to bear in understanding what the template is doing. More importantly, everything I know about QUEUEs can be brought to bear on making the tree behave as I wish it to.

The next most important fact is that "child levels are demand-loaded when a branch is expanded" (*Application Handbook*, p. 136). While wandering through the generated code, I discovered that child levels are also demand-*un*loaded when a branch is collapsed.

On first reading, however, I missed the fact that no secondary structure is referenced. This means that *all* levels or, more accurately, all visible levels are in a single QUEUE. Therefore, when any given node is expanded or contracted, the children at that level will be added to/deleted from the (single) QUEUE.

Lastly (and I *am* jumping ahead a bit), the QUEUE contains only a single field for the tree display and that field is a string. That is, regardless of what other fields are declared in the QUEUE structure, it is only a single string that will be displayed. This is easily inferred from the manual's instructions on how to format this string:

> Display String
>
> The field name or text to display for the primary file

level. This may be any valid Clarion expression, for example:

```
CLIP(CUST:LastName)&' '&CUST:FirstName
```

(*Appilcation Handbook*, p. 138). This direction only makes sense if the tree display is based on a single string.

This bit of intelligence is even more easily inferred from the fact that the documents state it rather explicitly: "The tree control is a single column list." Well, that makes it in pretty clear, doesn't it?

Gi' Me

In addition to relation tree standard behavior, there are some behaviors I would like to have. And there is at least one behavior I *require*.

I like trees where the parent level icon changes when the branch is expanded or contracted. I *am* aware that this is purely cosmetic but I like it and I want it. Therefore I *am* going to get it.

I need to have the Project record under the cursor "hot." That is, I need the last highlighted Project record to be current in memory. Why? Because this is what I need to know so that the appropriate project can be called and compiled.

Hot fields are a standard behavior in Browses. But relation trees are not browse-based. And, as there is no "Hot Fields" tab, this does not seem to be built into the template.

Taking a moment to consider my requirements, there are some things I clearly need to be looking for as I implement my relation tree control.

Cosmetics: Without even looking at the template prompts, I assume that there are prompts for conditional icon usage. I assume this because it is *so* typically Topspeed that I just cannot envision this feature being absent. It turns out that this *is* a safe assumption.

So I won't worry about icons. What I will need to watch out for is when a branch is expanded or contracted. I need to know how the template signals the expanded/contracted state or the change of state. Once I have that, the template will handle the icon change for me.

Business: Because neither the `QUEUE` entries nor the underlying file records are inherently hot when displayed, the parent record (in this case, a Projects record) is never loaded into memory for me. This is not really a problem because I can easily retrieve a `QUEUE` entry at will.

> *See? I told you that the fact that relation trees are `QUEUE` based was important! But I must also remind you that "Pride goeth before the fall" (if I didn't, you wouldn't continue reading, would you?).*

Having the `QUEUE` entry, if I include a unique key value in the display, I can parse out the key information and `FETCH` the entire record. Once I have the record, I can use the information in any way necessary.

Relation Tree Building Blocks

I won't walk you through the process of creating a relation tree. You are just as capable as I of following directions in the *Application Handbook*. You are also fully capable of replicating most of the silly mistakes I made when implementing a relation tree.

Instead, I'll concentrate on those aspects of the code needed to make my batch compiler work correctly. The most important item is that I need to know the project selected. Instead of popping up a "Select Project" browse, which would be inconsistent with the rest of the interface, I want to use the Project tree to select the Project by the position of the cursor in the list.

It is time to look at the QUEUE structure created by the templates. This is going to be one of those cases where I need to know what the template does, how it behaves and what it makes available. There is no knowledge of principles or methods to apply, deducing the required techniques. This time it is brute force; study the template, know what you're looking for, apply what is learned.

The QUEUE created by the relation tree template is:

```
Queue:RelTree     QUEUE,PRE()      ! Browsing Queue
REL1::Display        STRING(200)
REL1::Icon           SHORT
REL1::Level          LONG
REL1::Loaded         SHORT
REL1::Position       STRING(512)
                  END
```

Buried in the procedure's data section, the "! Browsing Queue" comment gives the queue away. (In this case, the template has created variables using the template instance number "1." This means that it is the first template populated. In another procedure, where there are already Code Templates populated, this number will be different.)

There are three very interesting looking variables here: REL1::Display, REL1::Level and REL1::Loaded. I already know what REL1::Position must mean and REL1::Icon, being a SHORT is either a flag or a pointer to an icon. In any case, these two don't seem to have much potential for the purposes at hand.

To trace the uses of the more interesting variables, I simply search for instances of each in the generated source and analyze the usage in the context of the code where I found it.

REL1::Display appears in routines where the display string is formatted. As one might have expected, REL1::Display is what the relation tree displays.

I know, therefore, that this is the variable that will end up containing the information I need to retrieve from the QUEUE. (You can also use REGET with the POSITION data stored in the queue.)

REL1::Level is very heavily used throughout the template. And it is tempting to give up the trace and simply assume that because the Projects file is the top level file, it is level 1.

Bad move.

During the expansion of the tree, the files in the schematic are read, from the top of the schema, down. A series of "Load" routines are called and in these routines, the display string is written to the QUEUE and levels are sequentially assigned.

If the top level of the tree is opened (expanded) and has children (subsidiary levels), it is assigned a value of 1. If it is at the top level but has no children, it is *also* assigned 1. However, if a level has subsidiary levels but is not opened, it is assigned -1 (actually, the documentation states that a negative value is assigned to any unexpanded level; see, pride *does* go before the fall). As the internal documentation within the template states, this flags that the node is expand*able*. So,

```
ABS(REL1::Level)
```

is safer for getting the tree level.

REL1::Loaded is assigned either True or False. If there are child records and they are loaded, it is assigned True. Otherwise, it is false. Obviously, this is an expanded/contracted flag. "True" means expanded, "False," contracted.

Fast and dirty, but there are the basics of how a relation tree operates. Now, to work.

Keeping The Project Current

The Projects file has two unique identifiers for each project. There is a project ID (though no one ever sees it) and there is a project name. Both are keys (solo or as the top node) and both require unique values.

The project ID would not be a meaningful piece of information for display but the project name would be. So, the project name will be placed in the tree. In fact at this level, the project name is really all that is required.

This means that if I have the QUEUE entry, I can do something like:

```
PRO:ProjectName = Clip(REL1::Display)
Access:Projects.Fetch(PRO:ProjectNameKey)
```

to get the entire project record, just as originally suspected.

And, because the tree uses a QUEUE, I can retrieve the underlying QUEUE record at any time easily enough:

```
Get(Queue:RelTree,Choice(?QueueFieldEquateLabel))
```

in the tree's NewSelection embed. And the entire code would look something like:

```
Get( Queue:RelTree , Choice(?QueueFieldEquateLabel))
PRO:ProjectName = Clip(REL1::Display)
Access:Projects.Fetch(PRO:ProjectNameKey)
```

However, what if the tree is expanded and the cursor is on an app name, not a project? If the highlighted line is a component application, *its* name is what GET() will return. This is hardly what is needed.

I do not want app names. So, instead of making every QUEUE entry "hot," I only want to GET() the QUEUE entry *if* the item is a Project name.

"All" I need to know whether the selector is on a Project. What I need to know, really, is which level of the tree the user is on. Then, I GET() the QUEUE entry only if the cursor is at the appropriate level (in this case, with only two levels, I am interested in level 1).

As my trip though the template demonstrated, REL1::Level contains the information I want in this case. I can confirm this by placing:

```
Message(ABS(REL1::Level))
```

in the NewSelection embed (after Parent call). So, my final code is:

```
If ABS(REL1::Level) = 1
  Get(Queue:RelTree,Choice(?RelTree))
  PRO:ProjectName = Clip(REL1::Display)
  Access:Projects.Fetch(PRO:ProjectNameKey)
End
```

If you are not comfortable with this, with the reliability of REL*x*::Level (and, during testing, I did manage to get a "2" where I expected a "1"), there is another way.

Consider the standard behavior of Access:*file*.Fetch(*key*).

Either Access:Projects.Fetch(PRO:ProjectNameKey)returns a value or it doesn't. If it doesn't, the FETCH was successful (FETCH() returned Level:Benign). If it returns *any* value, it failed. Failure means that whatever was assigned to PRO:ProjectName was *not* a project name (since there are only two levels, it must be an application name).

Therefore, if I define a datum,

```
SaveProject Like(PRO:ProjectName)
```

this code will serve exactly the same purpose:

```
Get(Queue:RelTree,Choice(?RelTree))
PRO:ProjectName = Clip(REL1::Display)
If NOT Access:Projects.Fetch(PRO:ProjectNameKey)
  SaveProject = PRO:ProjectName
End
PRO:ProjectName = SaveProject
```

If you are not 100% comfortable with the superficial backwardness of 0 (zero) meaning "Ok,"

```
Get(Queue:RelTree,Choice(?RelTree))
PRO:ProjectName = Clip(REL1::Display)
If Access:Projects.Fetch(PRO:ProjectNameKey) = Level:Benign
  SaveProject = PRO:ProjectName
End
PRO:ProjectName = SaveProject
```

has a more linear look and feel.

(I am not ignoring the possibility that an app and a project might have a common name. App file names are stored with an extension so duplication is not actually possible.)

Expanded Or Collapsed?

Now I know two different ways to tell whether the highlighted record is from a particular level of the tree. For the conditional icon display, I need to know how to determine whether a given level is expanded or collapsed.

Searching through the generated code, I found the `REL1::Load:Projects` routine.

It contains a `LOOP` testing that records exist in the child file and, if there are any, loading them into the `QUEUE`. When the `ROUTINE` actually adds an entry to the `QUEUE`, it also assigns

```
REL1::Loaded = True
```

There is also an `UnloadLevel` routine. In it, not only are the child entries deleted from the `QUEUE`,

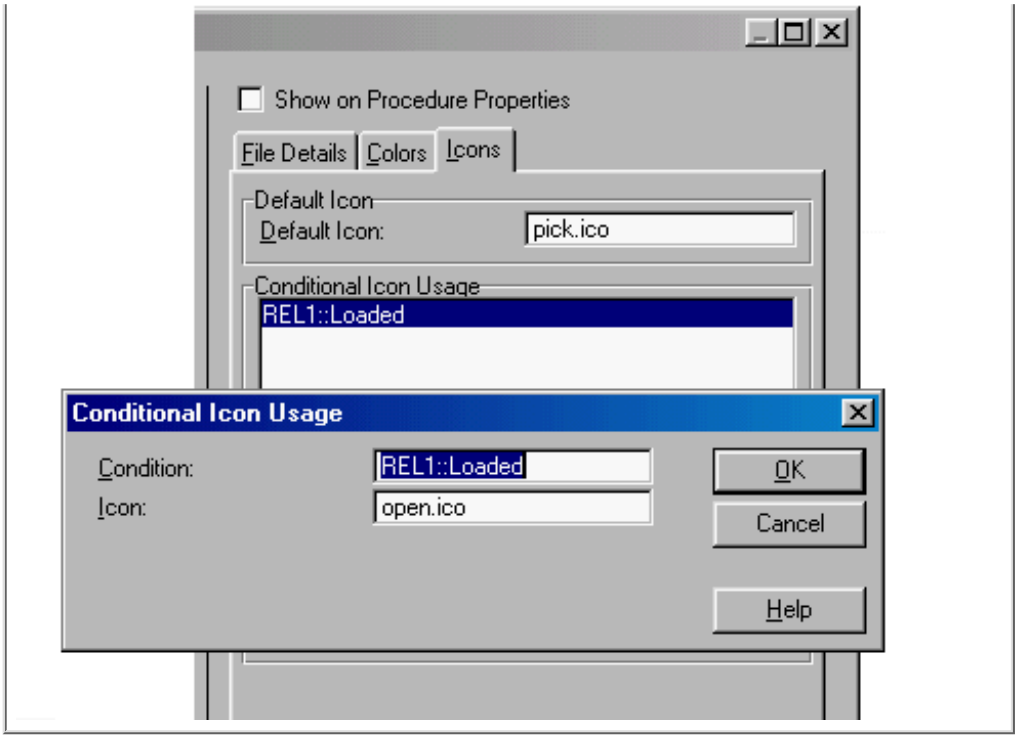
```
REL1::Loaded = False
```

is assigned.

So, the "flag" I am looking for *is* `REL1::Loaded`.

Sure enough, when `REL1::Loaded` is used to trigger the icons, different icons will be displayed when the branch is expanded or contracted (see Figure 4).

Figure 4. Setting up conditional icon usage.



(If you need to make the relation tree turn handflips, you owe it to yourself to check out Phillip Carroll's UltraTree template.)

Summary

"They're pretty, Colonel, but can they fight?" asked Donald Sutherland of Robert Ryan in "The Dirty Dozen." Now I have to adjust to the fact that I can only add a new project when the column header is highlighted (when a project is highlighted, I get the add app form). Perhaps I should declare the same update form for both levels? Maybe I shouldn't have an update at the application level?

A relation tree is attractive but can be confusing when it comes to maintenance. In this case, however, it works quite nicely. Yes, relation trees *can* fight and I don't have to mouse around as much as I once did.

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitor's right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Relation Trees:
A Few Of The Finer Points
(Nov 16,1999)

Presenting Many-To-Many Relationships:
Part 2
(Nov 16,1999)

The Clarion Advisor:
Breaking Out Of Nested Loops
(Nov 16,1999)

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Three Ways to Present
Many-To-Many Relationships
To The User

Part 2 of 2

by Tom Ruby

In the first article in this two-part series I explained some of the background behind many-to-many relationships, and discussed the "Form And Lookup" approach to presenting many-to-many data to the user. In this second part I cover the "Check List" and "Selection Pool" approaches.

> NOTE: The example application contains completed procedures. If you want to follow along with the example application, you can either create new procedures with different names, or create a new example application and import demo app procedures as required.

Check List

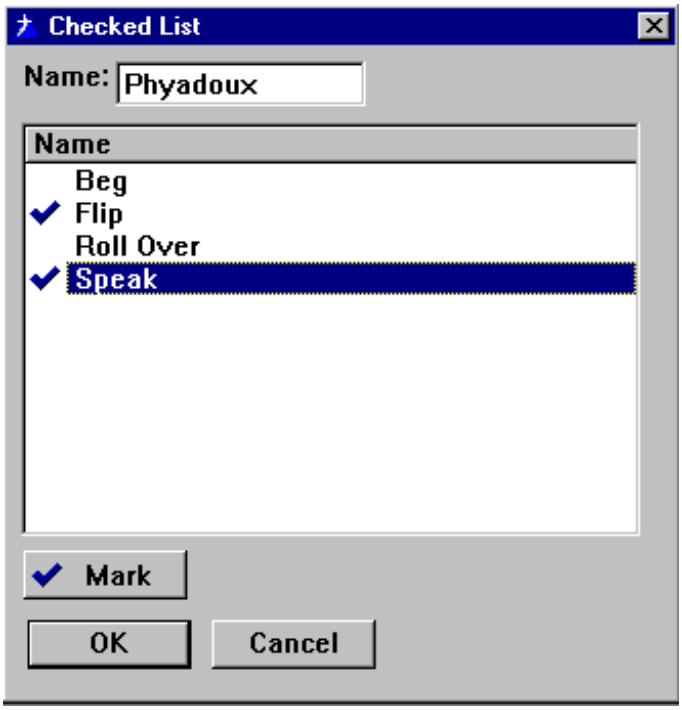
A check box list presents a list of all the options available and indicates which are selected (See figure 5). I use it when the user thinks of tagging which options apply or don't apply. This is useful when the list of options is fairly restricted, but would get unmanageable if the list contained hundreds or thousands of items. To try this out, make a new menu option on your frame to call a browse of puppies, and make a new form procedure for the browse update buttons. On the new form, populate the puppy name field and a browse box.

Figure 5. A check box list.

Relation Trees: A Few Of The Finer Points (Nov 16,1999)

Presenting Many-To-Many Relationships: Part 2 (Nov 16,1999)

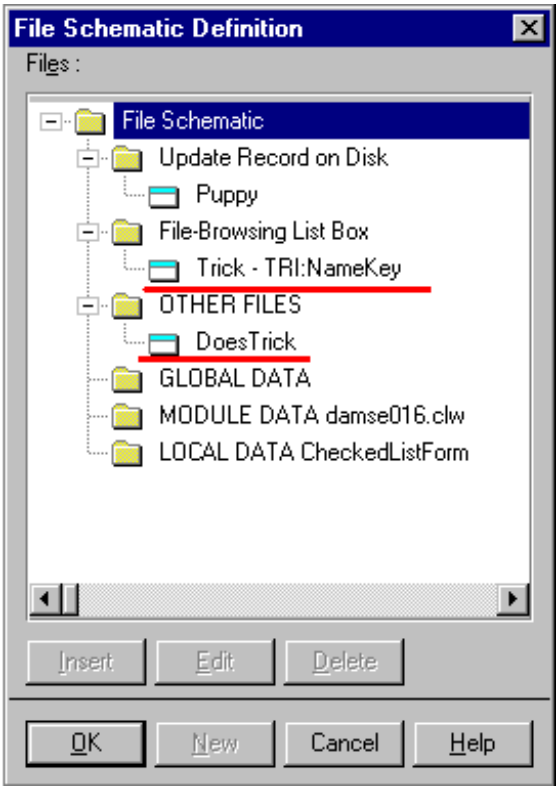
The Clarion Advisor: Breaking Out Of Nested Loops (Nov 16,1999)



In the file schematic for the browse box, select the Trick table and use the edit button to select the `TrickName` key so the tricks are listed alphabetically. In the list box formatter, add the Trick Name field, and select Normal under icons on the appearance tab. You won't need a range limit here since you want to show all the tricks.

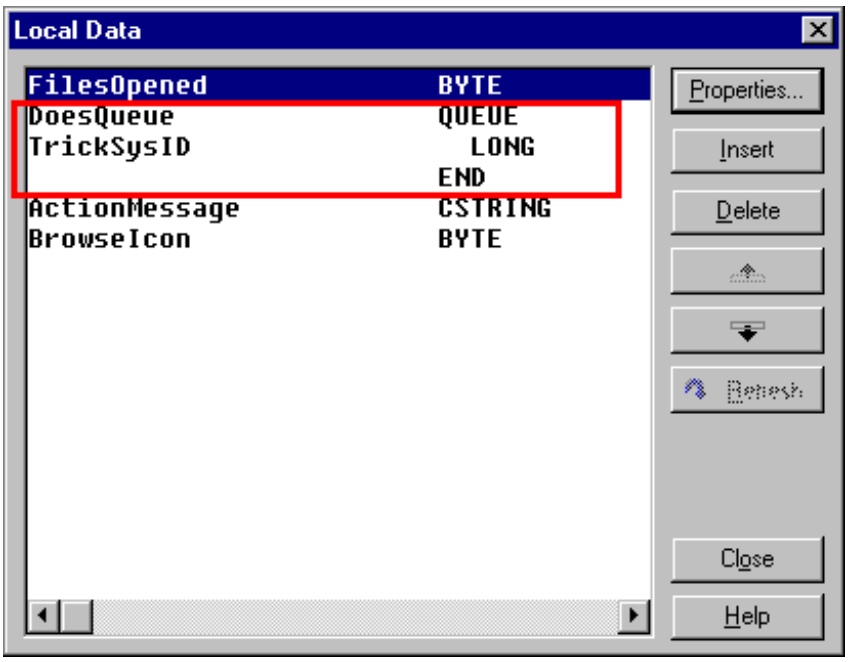
The browse box will need to pick which icon to show in its `SetQueueRecord` method, and this information is in the `DoesTrick` table, so go to the procedure's file schematic and add the `DoesTrick` table under Other Files (See figure 6). You can have the `SetQueueRecord` method look in the table to see if there is a `DoesTrick` record for each trick, but in many database environments this would be inefficient. Granted, it probably wouldn't matter in this application, but go ahead and do it the hard way just for fun.

Figure 6. File schematic for a check box list.



Next make a queue to hold the `TrickSysID` of all the tricks this puppy knows how to do, and refresh the queue when the browse is reset. I built the queue in the data area for the procedure using the data button and gave it a prefix of `DQ`. (See figure 7)

Figure 7. A queue in the procedure's data.

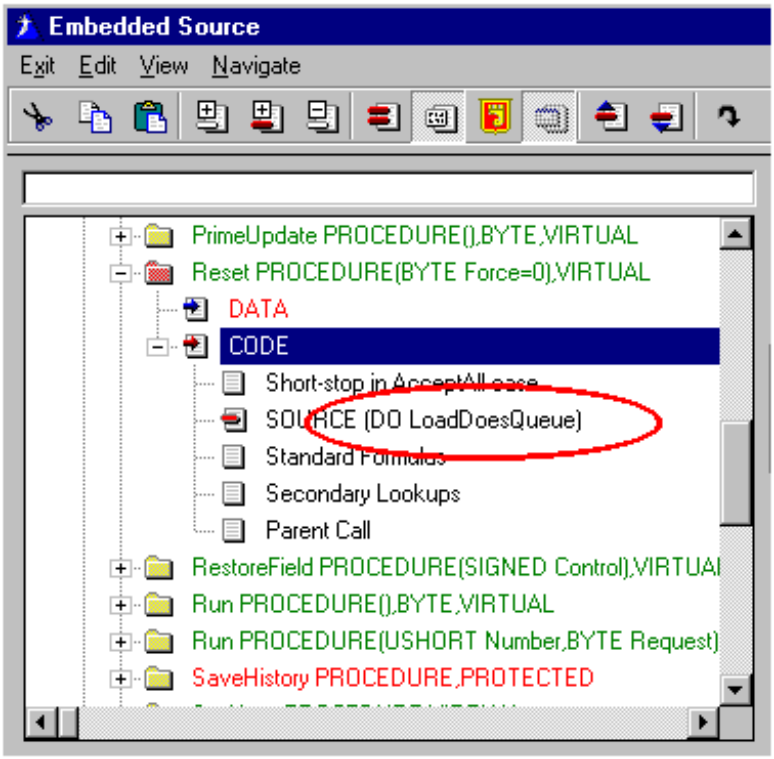


You will need some code to fill this queue. I put it in a routine:

```
LoadDoesQueue ROUTINE
  FREE( DoesQueue )
  Access:DoesTrick.ClearKey(DOS:PuppySysIDKey)
  DOS:PuppySysID = PUP:PuppySysID
  SET(DOS:PuppySysIDKey,DOS:PuppySysIDKey)
  LOOP
    IF Access:DoesTrick.Next() <> LEVEL:Benign OR !
    DOS:PuppySysID <> PUP:PuppySysID THEN BREAK .
    DQ:TrickSysID = DOS:TrickSysID
    ADD( DoesQueue )
  END
  SORT( DoesQueue, +DQ:TrickSysID )
```

Put a call to this routine in the `Reset` method of the window (See figure 8), and in the `ResetFromAsk` method of the browse.

Figure 8. The `ThisWindow.Reset` embed point.



You will need to add a variable to the procedure. I called it `BrowseIcon` and made it a byte. Go to the browse actions tab, and slide the tabs along till you find the Icons tab. Look at the properties for the `TRI:TrickName` icon. Press Insert to add a condition. The condition will be `BrowseIcon = 1`, and the icon will be whatever icon you want displayed for selected tricks. If you want to show an icon for unselected tricks, put that icon in for Default icon. Back out to the window formatter.

Next, put some code in the `SetQueueRecord` embed. Double click the browse box in the window formatter and find the Local Objects|BRW5|SetQueueRecord code embed, and press Insert. Then select Source. The embed code will look like this:

```
DQ:TrickSysID = TRI:TrickSysID
GET( DoesQueue, +DQ:TrickSysID )
IF ERRORCODE() OR DQ:TrickSysID <> TRI:TrickSysID
  BrowseIcon = 0
ELSE
  BrowseIcon = 1
END
```

You want this code *before* the generated code so the `BrowseIcon` variable will be set when the generated code looks at it to pick the

icon. Otherwise, the icon will indicate whether the trick above it is selected.

Penelope (remember Penelope?) will want to be able to turn the icons on and off, so put a set of browse update buttons on the form, and fill in a new procedure name. I called it CheckPuppyTrick. You don't want to insert or delete tricks here, just insert or delete DoesTrick records, so hide the insert and delete buttons. You will need to tell the browse template that there aren't any insert and delete buttons, so go to the embed editor and find where the BRW5.InsertControl variable is set (about priority 8505). In the next available embed, put:

```
BRW5.InsertControl = 0
BRW5.DeleteControl = 0
```

Now, make the form procedure. I used the form template and selected Trick as the file being updated. List Puppy and DoesTrick under Other Files. You'll need to press the Window button to make a window so the template will be happy, but don't worry about what it looks like because the user will never see it.

Go to the embed tree for this "form;" look at the Local Objects|ThisWindow|Init embed and find where the files are opened. Put an embed after the files are opened which looks like this:

```
IF SELF.Request <> ChangeRecord
  SELF.Response = RequestCancelled
  RETURN LEVEL:Fatal
END
DOS:PuppySysID = PUP:PuppySysID
DOS:TrickSysID = TRI:TrickSysID
IF Access:DoesTrick.Fetch(DOS:PuppySysIDKey) |
     = LEVEL:Benign
  Relate:DoesTrick.Delete(0)
ELSE
  DOS:PuppySysID = PUP:PuppySysID
  DOS:TrickSysID = TRI:TrickSysID
  Access:DoesTrick.Insert()
END
SELF.Response = RequestCompleted
RETURN LEVEL:Fatal
```

This embed code does four things. First, it checks that it isn't being asked to insert or delete a record and sets SELF.Response to RequestCancelled so any calling browse will think the user opted to cancel. Second, it checks to see if there is a DoesTrick record for the Puppy and Trick. Third, if there is a DoesTrick record it deletes it, and if not, it adds one. Fourth, it sets SELF.Response to RequestCompleted and returns LEVEL:Fatal.

The RETURN:LevelFatal statement will cause the form to close without ever displaying the window, and since SELF.Respose has been set to RequestCompleted, the Browse box which called it will think the user changed the record and do its ResetFromAsk method to refresh the display.

Notice that this form procedure doesn't care if it is called from a puppy form or from a trick form, so you can use the same procedure from a trick form.

To make check lists, remember to:

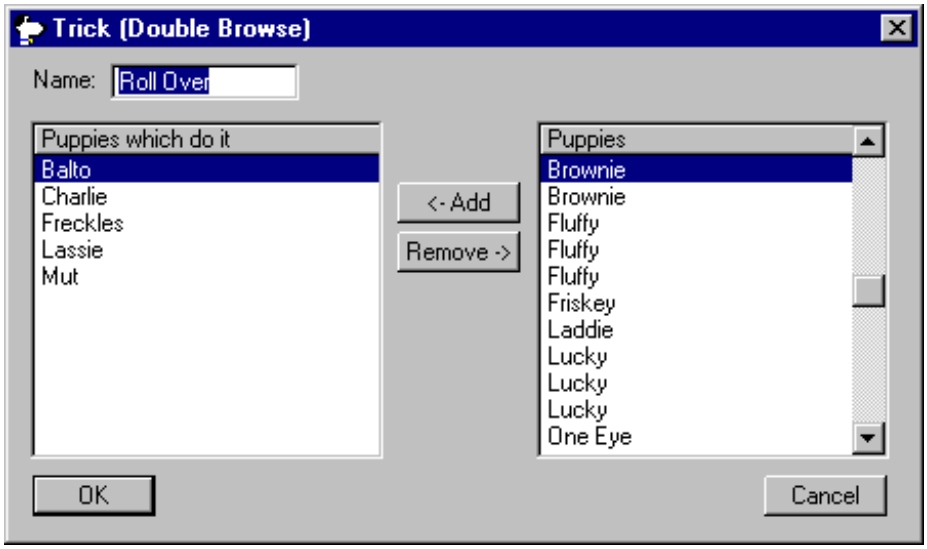
- Browse the other table and put an icon.
- Make the icon conditional on some local variable.
- Use the SetQueueRecord method to look in the cross reference table and set the local variable.
- Make a form to add or delete the cross reference records but don't let it open its window.

Selection Pool

A selection pool is useful where a user thinks of picking an option to

add. It shows two browses. One browse shows the selected options and the other shows the available options. This is useful when the user thinks about the relationship as "adding" one thing to the other.

Figure 9. A selection pool browse.

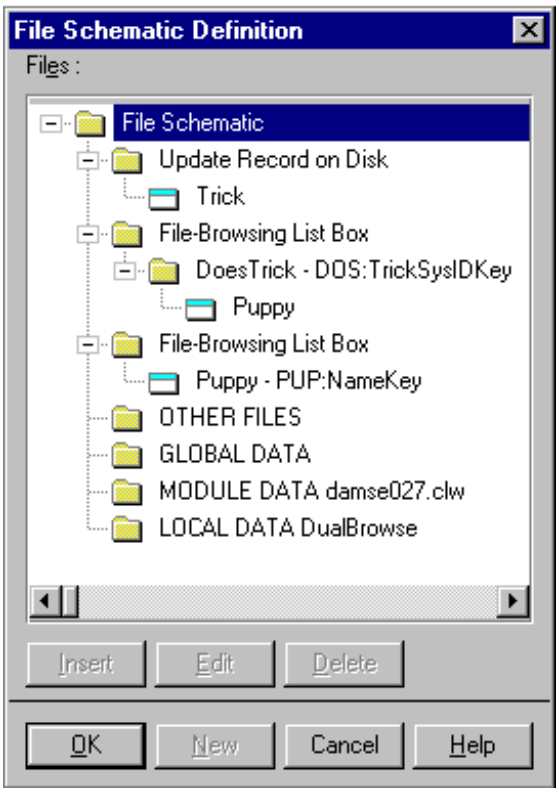


Make another option in your app frame menu to call up yet another browse, and make the browse show tricks. Make yet another trick update form, and place a control to edit the trick name. Make the form wide enough for two browses side by side with some space between them for buttons.

On the left side of the form, put a browse to show the `DoesTrick` table using the `TrickSysIDKey` key. Have it show the `PuppyName` out of the `Puppy` table. Range limit the browse on `TrickSysID` file related to the `Trick` table. Set an additional sort field to the puppy name so the puppies will be listed alphabetically.

Put another browse on the right side of the display to browse the puppy table. Yes, I know this table is used in the other browse, but don't worry, it will work fine (See Figure 10). Have it use the name key so the puppies will be listed alphabetically. You'll want to know the class names for these two browses, so look them up on their actions tabs and write them down or make them something meaningful. Mine are `BRW5` and `BRW6`.

Figure 10. File schematic for a double browse form.



Now, look at the procedure's embed tree and go to Procedure Routines. Build yourself two routines. I usually put them in two different embeds so they both show in the embed tree. They should look like this:

```
AddPuppy ROUTINE
   IF CHOICE(BRW6.ListControl)
       BRW6.UpdateBuffer()
       DOS:PuppySysID = PUP:PuppySysID
       DOS:TrickSysID = TRI:TrickSysID
       Access:DoesTrick.TryInsert()
       ThisWindow.Reset(1)
   END

RemovePuppy ROUTINE
   BRW5.UpdateBuffer()
   IF Access:DoesTrick.Fetch(DOS:DoesSysIDKey) |
         = LEVEL:Benign
      Relate:DoesTrick.Delete(0)
      ThisWindow.Reset(1)
   END
```

Take a close look at these two routines because they do the work.

`AddPuppy` first checks to see if a puppy is selected in BRW6. Second, it calls `BRW6.UpdateBuffer` so the puppy record will contain the selected puppy. Third, it builds a `DoesTrick` record and inserts it. The routine uses `TryInsert` instead of Insert so if the user mistakenly adds a puppy twice the program will appear to do nothing rather than showing a strange looking error message. Fourth, it resets the window.

`RemovePuppy` does an `UpdateBuffer` on `BRW5` so it knows which puppy it is removing. The browse is listing `DoesTrick` records, so `BRW5.UpdateBuffer()` will get the `DoesTrickSysID` into the `DoesTrick` record. It then fetches the `DoesTrick` record by the `SysIDKey`. Then it deletes the record and resets the window.

Now that you have these two routines, put `DO AddPuppy` in the embeds for the Add button (Control Events|?Button3|Accepted). Put `DO RemovePuppy` in the embeds for the Remove button.

Penelope can now add and remove puppies to her heart's content using these buttons. Since she's "moving" records from one list to another, this would be a natural place to use Drag and Drop, so go ahead and add drag and drop IDs to the list boxes. On the left side, the Drag Id will be RemovePuppy and the drop Id will be AddPuppy. On the right side, do just the opposite and make the Drag ID AddPuppy and the drop ID RemovePuppy. Double Click the browse on the left to get the embed tree and go to Control Events|?List|Drop, and add DO AddPuppy there. Double Click the browse on the right and go to Control Events|?List2|Drop, and add DO RemovePuppy there.

You may want to restrict the right side browse to show only the puppies that are absent from the left side browse. I haven't figured out how to filter a view based on records that aren't there, so I built a queue in the procedure data to store the SysID's of puppies that do the trick and used the browse's verify record embed to check it. Performance shouldn't be too terrible unless there are 200,000 puppies and only two are included in the list. I gave the queue a prefix of PQ. An embed in ThisWindow.Reset() method fills this queue:

```
FREE(PuppyQueue)
Access:DoesTrick.ClearKey(DOS:TrickSysIDKey)
DOS:TrickSysID = TRI:TrickSysID
SET(DOS:TrickSysIDKey,DOS:TrickSysIDKey)
LOOP
  IF Access:DoesTrick.Next() <> LEVEL:Benign THEN BREAK .
  IF DOS:TrickSysID <> TRI:TrickSysID THEN BREAK .
  PQ:SysID = DOS:PuppySysID
  ADD(PuppyQueue)
END
SORT(PuppyQueue,+PQ:SysID)
```

The embed to filter the browse looks like this:

```
PQ:SysID = PUP:PuppySysID
GET(PuppyQueue,+PQ:SysID)
IF ERRORCODE() OR PQ:SysID <> PUP:PuppySysID |
     THEN RETURN 0 .
RETURN 2
```

The code returns 0 if the puppy sys ID is not in the queue, and a 2 if it is. Be careful! If you return a 1, the browse class will decide this record is out of range and will not look any farther.

To show a selection pool, remember to:

- Browse the cross reference table displaying identifying fields from the other table.
- Browse the other table.
- Make Add and Remove routines to build and remove the cross reference records. These should reset the window when they're done.
- Make controls and/or drag & drop to call the Add and Remove routines.

Summary

There are many ways to present many-to-many relationships to the user, and if you're clever you can hide the cross reference from the user. The trick is to figure out how the *user* envisions the task, and make an interface accordingly. I hope you find these three approaches useful.

Downoad the example application

Tom Ruby, who is no relation to the man who shot Lee Harvey Oswald, is an independent contractor living in the middle of a hayfield in Central Illinois with his wife Susan and two red headed sons, Caleb and Ethan. He has been using Clarion for Windows since

the summer of '95. Before that, he was a "TopSpeeder" using Modula II, so he has never used the DOS versions of Clarion.

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

The Clarion Advisor: Breaking Out Of Nested Loops

by Dave Harms

If you write much Clarion code, you probably use `loop` structures a lot. Most likely you know about the various forms of `loop`, including `loop while`, and `loop until`. And you may have read that you can have your `while` and `until` statements at the bottom of the `loop` if you like.

A lesser-known feature is the ability to break out of nested loops. Consider the code in Figure 1.

Figure 1. A fairly pointless code fragment with a problem.

```
   loop
      x = random(1,999)
      loop 100 times
         y = random(1,999)
         display()
         if y = x then break.
      end
   end
```

All the code in Figure 1 does is set up two loops, an inner and outer loop. One variable is randomly set in the outer loop, and another in the inner loop. The inner loop has 100 tries to match its value with the outer loop value before the outer loop value is reset.

But there's a problem. The code in Figure 1 is going to run forever, because the `break` statement only terminates the inner loop, while the outer loop continues (once again invoking the inner loop).

Traditionally there have been two ways to deal with this problem. One is to set up a local variable which you test for at each loop level, and exit if the flag has been set. This is workable but awkward, particularly when there are a number of nested loops.

The other solution is to employ a `goto` which forces execution to jump to the specified label. The use of `goto` is generally frowned upon as it tends to make code harder to read.

A better alternative to these approaches is to use loop labels and a parameterized `break` statement, as shown in Figure 2.

Figure 2. The code fragment with a loop label.

```
   open(window)
l1 loop
      x = random(1,999)
      loop 100 times
         y = random(1,999)
         display()
         if y = x then break l1.
      end
   end
```

Note that the code now has a label in the first column of the first loop

Relation Trees: A Few Of The Finer Points (Nov 16,1999)

Presenting Many-To-Many Relationships: Part 2 (Nov 16,1999)

The Clarion Advisor: Breaking Out Of Nested Loops (Nov 16,1999)

statement. Inside the loop is the code

```
break l1
```

which forces a break in the outer loop whenever matching numbers are found, enabling a clean exit with a minimum of coding.

This is a simple example, but you might also have a more complex set of nested loops to deal with. In that case you can put labels on each of your loops, and break out of specified levels as your program logic dictates. You can even use this form of the break statement in an `accept` loop.

Download the source code

Relation Trees:
A Few Of The Finer Points
(Nov 16,1999)

Presenting Many-To-Many Relationships:
Part 2
(Nov 16,1999)

The Clarion Advisor:
Breaking Out Of Nested Loops
(Nov 16,1999)

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

ABC Design: The ViewManager

by David Bayliss

Part 1 of 2

Having dealt with the `FileManager` and `RelationManager` classes the final part of the file system I need to deal with is the `ViewManager`. It could be argued that having three different objects dealing with files is a little excessive. In a sense this is true; logically the `ViewManager` brings fairly little to the table over a `RelationManager` (it "just" deals with a bunch of files). However, the `ViewManager` does handle the very important special case where a bunch of files are being used to retrieve a series of records (including child lookups) in a nominated sequence. This was so vital I felt it deserved a special object. It turns out that the `ViewManager` is almost never used as a `ViewManager`, but it *is* the base class used for many of the higher level data access objects.

Room With A View

The `ViewManager` is to a `View` structure what a `FileManager` is to a `File` structure. Specifically, the `ViewManager` is there to act as an OOP front end to the view. It is also there to provide some logical sophistication not present in the native view. The main logical elements brought to the table are:

. Range Limits: The ABC and legacy template chains have the notion of a range limit, which is the ability to restrict the records retrieved to those matching one or more of the major-most elements of the key order. Tied in with the notion of range limits is the notion of a free element (the major-most element of a sort order that is not range limited).

b. Multiple Sort Orders: A Clarion view structure only supports a single sort order (although the current sort order can be changed at will). The `ViewManager` is to support multiple sort orders (simply).

c. Flexible filters: The Clarion view structure has one assignable filter. The `ViewManager` provides for multiple filters active at once.

d. Attitude: A key requirement of ABC was that we wanted optimal (or near optimal) browse performance. It was felt that an essential element of that was ensuring that views were handled cleverly. Rather than us having to build the *cleverness* into every object that used the view (browse, drop down list, drop down combo, report, process etc) the `ViewManager` watches the commands being sent to the view structure and translates these commands (where required) into a more intelligent sequence.

Initialisation

The initialisation section of the `ViewManager` is quite large, and it also offends the OOP purists as it contains state; the order in which the methods are called is significant. These two are tied together. A significant procedure (say five browses and 15 drop combos) will contain 20 view initialisation sequences and possibly 100 sort order creation sequences. We felt that having these sequences concise,

readable and efficient was more important than sheer hygiene.

The call sequence is:

```
Init
[ Repeat 1 or more times
        AddSortOrder
        AppendOrder     ! Optional
        AddRange        ! Optional
]
UseView
```

AddRange PROCEDURE(*? Field)
AddRange PROCEDURE(*? Field,*? Limit)
AddRange PROCEDURE(*? Field,*? Low,*? High)

The three AddRange methods set a range limit upon the current sort order (defined by the preceding AddSortOrder or SetOrder). They correspond to Current Value, Single Value and High/Low Value range limits respectively. The code for each is similar. The aim of the code is to produce a RangeLimit queue with a queue record defined for each element of the key that is range limited.

The first line of code sets the type of the range limit. The second calls LimitMajorComponents. It is defined in ABC that if the range-limited field of a sort order is not the most major component then all the more major components are implicitly current-value limited irrespective of the limit-type of the more minor component. The call to LimitMajorComponents implements this detail.

The field(s) passed in is then added to the range-limit queue. Finally SetFreeElement is called to compute the free element of the sort order.

AddRange PROCEDURE(*? Field,RelationManager↵
MyFile,RelationManager RelatedFile)

The purpose of this AddRange is the same as the other three, but the code is somewhat different because the information required to construct the range limit is not publicly available (it is hidden in the RelationManager). Essentially if two files are related by keys with, say, three components, then a FileLimit range limit corresponds to a single-value limit upon three different elements! The queue is thus filled in using the ListLinkingFields capability of the RelationManager.

AddSortOrder PROCEDURE(<KEY K>),BYTE,PROC

The AddSortOrder method actually performs the action of creating a new logical sort order. Each logical sort order can have a different range limit (and thus free element), and a different filter. This is stored in a queue (or Order queue) with a record pertaining to each sort order.

This method clears the queue record (it has to as it contains an ANY), stores the key, creates a new range-limit list and uses the first component of the key as the free element.

The sort order itself is computed by passing the key (if present) as a comma delimited list of components to the SetOrder method.

As it is impossible to remove sort orders it is okay to return the record number of the queue record added as the "unique identifier" of the sort order within the queue (for later use by SetOrder).

AppendOrder PROCEDURE(STRING Order)

Each time I look at the AddSortOrder/AppendOrder relationship I oscillate between deciding it is a beautifully elegant engineering solution and fretting that it is a complete hack. The issue is this: *Proper* database theory will tell you that sort orders should be defined in terms of fields and ascending/descending flags so that the logic of the program is nice and clean, and the ugliness of physical database design and actually getting performance out of the system can be left to some other poor schmuck.

The problem is that with many programmers coding furiously and requiring many different sort orders, the "poor schmuck" (typically the DBA) actually may not be able to get the system to perform at all! Worse yet he may be sufficiently senior that you can't boss him around to make sure *your* program runs ok.

Thus the programmers have to get pragmatic and they start building keys into their program logic. Performance improves greatly; unfortunately applications have to use a restricted set of sort sequences or the number of keys explodes.

So along comes ABC. What do we do, "restrictive" or "slow"? Legacy took the restrictive approach, which I didn't want, but the alternative wasn't that nice either. What we settled on in ABC was an interface that *can* be used in a fully, logically pure way but that *encourages* the writing of efficient queries. It does this by defining a sort order in two parts. The first (optional) part is a key which defines the main part of the sort sequence. The second (also optional) part is the fields used to sort duplicates within the first key. Combining this with some special technology within the view driver we have the panacea of full flexibility that is usually fast.

To put some meat on the bones; assume you have an invoice file containing a customer id, and an invoice date (amongst other things). There is a key on `customerid`. You want to list invoices in customer order with invoices for a customer listed in date order. You do an `AddSortOrder(CustomerKey)` and then an `AppendOrder('INV:Dateordered')`. The view driver will then read in the records for the first customer, sort them, then the second customer etc. Given that sorting is at least an NlogN process this "bucket sorting" can produce a massive time savings.

An additional tweak that should be available by C6 is that AppendOrder may start with a "*" character, meaning the specified order replaces the order specified by the key *after and including* the free element. This is useful for specifying range-limit information using a key but then ignoring any trailing key components.

```
Init PROCEDURE(VIEW V,RelationManager↵
  RM,<SortOrder SO>)
```

Much of the init code is straightforward; note though that the order property may be set up in one of two ways. If passed in then that version is used, otherwise one is created. This allows a derived class to construct a larger queue (more fields) than the `ViewManager` requires while still having the `ViewManager` perform the administration required.

The order queue is central to the whole `ViewManager` operation and stores all the information pertaining to a given sort order. When the `ViewManager` is derived by a browse it stores all the information for one tab of the browse.

Note also the default values provided to the view driver to enable the SQL buffering technology.

The Init method also ensures the `UseView` method is called, for reasons I'll discuss under that method name.

```
Kill PROCEDURE,VIRTUAL
```

This is one of those messy little procedures that only really exists to ensure that there aren't any memory leaks.

Essentially `Kill` just loops through the records of the order queue, and for each element of that queue it cycles through each element of the filter queue freeing up each filter element. Then it disposes of the filter and order-clause queue and nulls out the free element (which is an `any`).

Finally if the order queue needs freeing (meaning it wasn't passed in from outside) then it's disposed of. Finally the order queue is freed.

```
UseView PROCEDURE,PROTECTED
```

UseView has a simple task, to call UseFile on all the files a view references. (UseFile technology is explained fully in my article on the FileManager class.) The requirement of calling UseFile is a little subtle.

Logically when the template uses a view (or a browse) it only knows about the primary file: the lookups are an implicit part of view functionality. But the implementation of the VIEW (within the Clarion language) has one or two legacy throwbacks. One of these is that the files have to be opened independently before the view will work. So the ViewManager "dresses" the view structure to tidy this up, making sure all of the underlying files have been used at an ABC level and are therefore likely to be open, thus satisfying the view.

A list of file references is available from the view driver itself. The FileManager reference is obtained from the internal lists so that UseFile can be called.

Summary

That takes care of the initialization and cleanup code. In Part II I'll look at the ViewManager methods used in typical browse operations.

David Bayliss is a Software Development Manager for Topspeed Corporation. He is also Topspeed's compiler writer and the chief architect of the Application Builder Classes.

ABC Design Series: The ViewManager Part 1
(Nov 23,1999)

November 1999 News
(Nov 23,1999)

The Art Of Software Development: Analysis By Design
(Nov 23,1999)

Clarion Magazine Version 2!
(Nov 23,1999)

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

Clarion News

November 23, 1999

Orbtech On Clarion
Patrick O'Brien has set up a Clarion section on the Orbtech web site. Feature pages so far include an introduction to Clarion and a list of Clarion resources.

Data Modeller 5.5 Now Available
Data Modeller 5.5 is now available for Clarion 5 EE and Clarion 5 PE users. The upgrade price is US$49. New features include an entity design mode, logical and wizatron views, an MS-SQL7 script writer, enhanced test data generator, and a query builder.

Send "Why Clarion" To A Friend
TopSpeed has a web page that lets you send the "Why Clarion" article to someone's email address with a click of a mouse.

Clarion At Harley-Davidson
Harley-Davidson has been a long-time user of Clarion development tools. This Clarion success story is available at TopSpeed's web site.

Enhanced Version Of Super Security
Version 4.50 of Boxsoft Development's Super Security Template is now available for download. New features include Clarion 5.5 beta support, sharing of global security settings across applications, multi-session user name storage, user name hiding and password change on the logon window, caching of CheckAccess calls, and more. This is the final version with Clarion 4 support. This new version of Super Security is available through the end of November for the current price of $149. On December 1, 1999 the price increases to $179. Upgrade pricing is available for current users.

List & Label Templates Demonstration
Simon Burrows has set up a web site with a demonstration of the new List & Label templates. List & Label is a third party reporting product from Combit Software.

Free Enter/Tab Solution
solid.software has released EnTabber, a system utility to allow the enter key to be used like a tab key. EnTabber installs a system wide keyboard hook which can be turned off and on from the system tray with a single mouse click.

Free Clarion Contacts Manager With $100 Purchase
Sterling Data is offering its Contacts Manager free with any purchase of Sterling templates worth $100 or more. The Sterling Data Contacts Manager includes built-in Word-like word processing (High Edit VBX) and a variety of ideas and tips for your own applications.

CWCM Free Compile-Manager Update
A new version of CWCM Compile-Manager is now available for download. New features include getting procedures from apps, loading app and jumping to a procedure (C5EE), and finding procedures by template, name, or prototype.

November 16, 1999

ABC Design Series: The ViewManager Part 1
(Nov 23,1999)

November 1999 News
(Nov 23,1999)

The Art Of Software Development: Analysis By Design
(Nov 23,1999)

Clarion Magazine Version 2!
(Nov 23,1999)

TX Text Control Class Wrapper Demo

A demo of the TX Text Control (www.textcontrol.com) OCX class wrapper is now available. This control with wrapper allows Clarion programs to view, edit and print .RTF, .DOC, .HTML and native .TX files.

TopSpeed Introduces New Third Party Program

TopSpeed has replaced its Third Party Program. Third Party Partners now receive a full page ad per enrolled product in the annual catalog, which ships with Clarion; one exhibitor's table at the Florida DevCon; access to alpha builds; private newsgroup; and access to TopSpeed authorized international distributors to negotiate their own agreements. Cost is $2000 per year per product.

Multi-Language PDFs

TopSpeed's now has the At-A-Glance PDF datasheets available in French, Italian, Spanish, and Portuguese as well as English. Choose the language from the drop down list before downloading.

TopSpeed Banners And Animated GIFs

TopSpeed banners and animated GIFs are now available for download. These images are intended for use as links to TopSpeed.com.

New Utility Eases Development

ProDomus has a new version log and command line utility that can be added to the project to execute at the end of compiles, or to the Accessories menu. It can be used to increment version or build numbers, store notes, or alter the Clarion environment (i.e. change your include directory). For C5 and C5.5.

Linder SetupBuilder 3.0 Beta 6 Patch 1

A new Linder SetupBuilder 3.0 (Beta 6 PATCH 1) is available now. This patch requires the Beta 6 release 6007 of SetupBuilder. New features include True Type Font installation (32 bit only) and custom serial number authentication. SetupBuilder 3.0 lists for $119.00 USD.

New Calendar And Animation Controls

New from Solid Software are a drop down calendar written in Clarion and a wrapper class for the Win32 animation control.

Clarion Web Browser

Richard Rogers has released a free, experimental version of his Prophecy web browser/desktop control. More to come, including modules and themes. Requires ABC.

Tintools Update For Clarion 5.5

A version of Tintools for Clarion 5.5 is now available for download. Tintools is a collection of freeware functions, procedures, viewers and templates.

CapeSoft Product Updates

All CapeSoft products, including the bundles, are now available from www.clarionshop.com. Most products now have C5.5 B1 compatible releases, with the rest to follow shortly. Minor upgrades include File Manager 2.9, EzHelp 2.1, and Multi-Proj 2.1. NetTalk, a product that enables inter-computer communication, is now in late alpha testing. Interested beta testers should contact beta@capesoft.com.

November 9, 1999

TopSpeed Drops Unpopular Registration Scheme

TopSpeed's Bob Zaunere has announced the company is dropping the registration scheme implemented in Clarion 5.5 Beta 1. The controversial registration process required developers to obtain a machine-specific unlock key either online or by telephone from TopSpeed.

East Tennessee Clarion Conference Slated For May 2000

The etc 2000 conference will be held May 24-26, 2000 in Gatlinburg, with pre- and post-conference classes on the 23rd and 27th. The location is again the Edgewater Hotel which has just completed an extensive remodeling of all rooms including the conference rooms. General session attendance (including most meals) is US$395.

Details to follow.

Clarion 5.5 Beta Newsgroups

Topspeed has added three newsgroups to tsnews.clarion.com: topspeed.beta.c55, for the 5.5 IDE and general functionality; topspeed.beta.c55.web, for issues related to Web functions; and topspeed.beta.c55.suggestions.

Clarion 5.5 Compatibility List On TopSpeed Turnpike

Tom Ruby has added a section to his justly famous TopSpeed Turnpike to track which third party products have 5.5 compatible versions of their products available. Updated products include CPCS, Arco Word Reporter, GNotes, GRegPlus, Linder Software products, ProDomus products, Sterling Data components and templates, and XLIB.

UK Developers - BT Big Number Source Code Available

BigNum from Sterling Data is an app to convert your customers phone/fax number area codes to the new ones being introduced by BT. Full source code is supplied. You just slot in your own phone & fax fields and compile it as a standalone EXE. The converter allows for area code changes which can be done now and local number changes which must not be done before 22/4/2000

New Calendar Control

Leonid Chudakov, Uncertified Clarion Developer, has posted another common control for Clarion developers. This month calendar control is a pure Clarion solution.

UltraTree Platinum Replaces Professional Edition

Paragon Design & Development has replaced its UltraTree Professional Edition product with the new UltraTree Platinum. Support for the Professional Edition will be discontinued after December, 1999. UltraTree Platinum has all of the features of UltraTree Professional, plus many new features, plus enhancements to many existing features. See the Paragon web site for full details and pricing, as well as information on new support packages.

November 2, 1999

Clarion 5.5 Beta Released!

Clarion 5.5 has gone into beta, with some developers already receiving their disks. This release offers major enhancements in the IDE, and integrates new Web technology into both the Enterprise and Professional editions. This week's issue of Clarion Magazine also contains a product preview.

Play AVI Files On Clarion Window

Leonid Chudakov has a new multimedia demo which shows how to play AVI files on a Clarion window.

Steven Gallafent's DevCon Examples Available

Steven Gallafent has posted the examples from his DevCon presentation, including an example program that sends an email message using the Catalyst SMTP DLL.

Stealth Software Modulizer Template Update

For registered owners of the Modulizer Templates there is an updated C5B version available. To get it email stealthsoft@mweb.co.za. The free working shareware demo is still available for download.

Free CopyFlash Offer

Sterling Data is offering a buy one, get one free deal. Buy any Sterling Data template (IMPEX,SearchFlash, BackFlash or LogFlash) before November 10, 1999 and receive CopyFlash FREE.

Drag/Drop Editing Template

Sterling Data has received a template from John Morter of Flat Chat Solutions (Australia) which is a variation on the freeware TrashFlash template. It allows drag and drop changes and deletes by using the standard browse buttons as drop zones.

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 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

The Art Of Software Development: Analysis And Design

by Bruce Gilham

Editor's Note: This is a somewhat delayed second part to an series Bruce Gilham began this summer on the art of software development. I'm saddened to report that Bruce had been in ill health, and passed away this fall.

Newer Clarion developers may not have heard of Bruce; others will remember him for the Los Angeles DevCons he sponsored earlier this decade. Perhaps there were other regional Clarion conferences before those in L.A., but I don't know of any. I do know that I made many new friends there, including Bruce. I'm grateful for the opportunity he afforded me to give my first conference presentation, and for his contributions to the Clarion community.

Dave Harms

This article picks up where Part One left off. Last seen, our hero was madly eliminating bugs. But correction just leads to more correction. Forward progress comes from creating something new. This brings us to the twin topics of design and analysis.

Rule #4: Analysis is cheaper than design; design is cheaper than programming. The very worst decision a company can make is to speed up a project by skipping the analysis phase or design phase. Prototypes not withstanding, the answer to any complaints about late delivery is more design, not more programming. The solution to slow or endless design is more analysis.

Analysis and design consist mostly of meetings with users and compilation of notes. Programming probably contributes *less* to the process of design than does playing video games. At least the time spent playing video games won't send the project off in the wrong direction. Analysis meetings are mostly with managers and some key users. Design meetings are usually with working managers and the more knowledgeable users. Seat-of-the-pants programming is not only *non*-productive, it can actually be *counter*-productive.

Analysis means breaking the project up into parts and looking at the relationships of those parts. The key activity in analysis is *naming the parts of the system!* Any part of a system that is not named, or that is given a confusing or inappropriate name, is bound to slow things down later on.

I recall a day-long meeting to sort out the meaning of "due date." At the end of the meeting we realized that due date was not the same thing for sales, shipping and production control. So every time we "fixed" due dates we got new complaints! This could have been completely avoided if we had investigated the various meanings of due dates in the beginning.

Almost all human communication is with words. Anyone without a vocabulary is isolated. It can be demonstrated conclusively that a student, in passing over a single misunderstood or not-understood

ABC Design Series: The ViewManager Part 1
(Nov 23,1999)

November 1999 News
(Nov 23,1999)

The Art Of Software Development: Analysis By Design
(Nov 23,1999)

Clarion Magazine Version 2!
(Nov 23,1999)

word, suffers immediate physiological changes. For example, the Learning Accelerator from Applied Scholastics monitors a student's non-comprehension by metering changes in the body's electrical resistance. Variances are instant and significant when a student passes over a word he or she does not understand.

There are mysterious words of all flavors. For example, I ask you to put a dollop of salsa on my taco, and not understanding dollop you put a wallop, and so I figure that you have it in for me. Or I tell you to "run up the temperature" of an oven and so you back across the room and run up to the machine to change the temperature.

Laugh if you will, but mysterious and misunderstood words are the bane of any programming project.

Rule #5: The analysis is done when all the parts of the system, extant and proposed, are named and have a defined purpose and responsible party or parties. The analysis document should include what kind of thing it is (program, report, screen, procedure, feature, business rule, database, field, etc.) and who (plural) has authority over it. This will usually be the most senior user. It also shows the purpose of the item in the context of the business.

The list must be complete and contain *everything* that could possibly be included in the system, immediately or in the future. While this sounds as if it would take a long time, remember that all you are interested in at this time is the name, purpose and authority. The decision to include it, delay it or abandon it comes later.

Finally, the analysis includes any imperative design issues. These are often expressed in fairly general terms, such as the requirement that an accounting system *must* be accurate. While a finicky or ugly accounting system would not be popular, and there are circumstances when it would be acceptable, an inaccurate one is never acceptable.

Also a glossary of terms is a good idea. If this is missing, any ambiguous names or terms must be defined in the text.

You might think of the analysis document as a shopping list.

> Note: Some developers expand the analysis stage to include all the questions that need to be asked of the user. In contrast, I assume that the users will have input at all stages.

The analysis phase naturally leads into the design phase. From a user, or system specifier's viewpoint, there is little difference between analysis and design; in both cases they still answer a lot of questions.

From the developer's viewpoint design is a new ball game. The analysis is general and all the parts of the system must be considered. In the design stage, if the analysis was adequate, the system is broken up into pieces and the focus can be on one or another of these pieces, not the whole system. When the developer finds excessive interaction between the parts of the system then more analysis is needed.

But what if the users refuse to invest the time for a full analysis? In this case the programmer is advised to covertly execute the analysis anyway. It's now a matter of doing so a bit at a time, or do what analysis that you can, then work out some sort of design (even if terrible) then finally write a "draft of the program." Tell the users that it is the draft version.

When you release the draft version of the software, and the quality is there, the users will answer the original analysis question. Never mind that they act like innocent victims of the big-bad-programmer. Simply point out that you're not a mind reader, and insist (politely) that they share with you the names of the parts of the system. This is a piecemeal approach, but it works!

With well-grooved clients the analysis phase is fast and exciting. Remember that everything worthwhile starts with a decision.

Rule #6: The end of the design phase occurs when all the users have no significant questions or objections. By significant I mean an issue that affects the data structure or business rules. In this day of web sites, it also means anything that would affect a majority of the pages.

What usually happens at the end of the design phase is that the users start changing things back to a recent version or all the changes they request are cosmetic. Only at this point is it really safe to invest in programming time. Prior to this any programming is prototyping or proof-of-concept and may not only be worthless, but might well be misleading.

All of us have come up with a great idea that proved impossible to implement. If the feature was promised but never seen, the disappointment is a lot less sharp then when a feature is "tested" but dropped. The answer is to use some intelligence when making a demo or proof-of-concept program and stick to tried and true features!

The largest danger comes from believing features lists for programming tools, or perhaps more seriously, the implied features. "Fast" sometimes means "hard to use" or "starved for features." "Advanced features" probably means "buggy." And so forth. It's one thing to kid the end user, but something completely different to mislead a programmer who makes his living form the tools he purchases. I'm not saying that you shouldn't use fast or feature-rich programming tools, only that you *must* know how the tools work *in production* before betting the project on them.

In general the design is done on paper and includes pictures of the screens and reports that will be supported. Where a screen is required to perform a complex operation a prototype is appropriate to test user reaction. Usually, but not always, a prototype is a substitute for adequate analysis. Only the most sophisticated of users can exhaust the utility of the simplest of screens.

Of course, users naturally have their brightest ideas when they see the actual product. This differs from an almost compulsive need to change things, change things, change things. A lot of learning takes place in the design phase. It might as well be called the *education* phase, as that is as important as having a final document.

When the client realizes that the program is "just what we asked for but not what we really wanted" the analysis and design need to be redone. At this point the compulsion to change is even stronger. If you don't redo the design phase, then when they see the program their immediate reaction, even if not voiced, is to want to change it, even though you already have. This can actually spin the project out of control with endless changes and bright ideas. Nothing reassures a client more than seeing his, or her, ideas on paper. The closer to his, or her, original concept the product is, the more reassured the client will be.

One of the most complex designs that I ever encountered was for a retail store. We wrote the system three times, and it was rejected three times. After working on site to get the total picture, I discovered that the client was in serious trouble for evading sales tax. Everything then became clear. They wanted to system designed in such a way that they could cheat on their taxes, but they were careful not to reveal the fact. It doesn't matter what I would have done had I known that they were cooking the books, but they were afraid of my finding out.

This brings up the next rule:

Rule #7: It is almost impossible to write a system for an organization engaged in criminal activities. The reader might think that this is obvious, but when you are deep in the design and nothing aligns, you automatically wonder what kind of criminal or unethical actions the client is trying to hide. A great number of companies have their dirty little secrets, but when the effort to hide the misdeeds outweighs the desire to have a clean and effective computer system, then the programmer is in trouble. Most likely he

will be blamed for everything and anything. He will be responsible for the current business conditions, the pot holes in the parking lot and the flicker on the monitor screen. Worst of all, the client will delay or refuse to pay for your hard work!

At this point it is vital to realize that it is not *your* bad performance, but the client's efforts to keep anyone from "finding out." If you have done your level best and find yourself a target, your first action would be to find out what it is they think you know.

Once I discovered the retail store's dirty little secret, they calmed down and we finished the system in a few week. And this was a system that had been through three major rewrites in nine months. Luckily I was not confronted by a moral dilemma—the sales tax people were on to them with a quarter million dollar fine. Good design would have revealed the true situation.

A less extreme cause of delays is that:

Rule #8: Every uncertainty about a system doubles the complexity of the affected feature. The programmer depends on the certainty of the specifications. Changes to the specifications, no matter how "insignificant" or "justified," will exponentially increase the development time. It can be a matter of pride among programmers as to how many changes they can tolerate before they lose it.

One could define a computer system as the collection of uncertainties of an organization, the maybes of the operation. You have an invoicing program because you don't know the details of the next invoice. If all invoices went to the same address for the same amount, then an invoicing program would be pointless. The "size" of the program expands rapidly with the so-called flexibility. A few well-chosen executive decisions can cut down the complexity of a bogged system and put it back on the time line. Unfortunately, decisions that are later reversed are worse then no decision in the first place, therefore such decisions are best made in the analysis phase.

Note that maintainability is not the same as changeability. Maintainability means "the ease of making changes to the system that parallel the changes in the operation of the business." These are usually minor in scope and hopefully the impact on the system is evaluated before the change is made. It is the difference between redecorating your apartment and changing the floor plan after the foundation is poured.

A changeable program would easily respond to every client whim, which is pretty much an impossibility. Writing a really changeable program has been described as "nailing Jell-O to a tree."

This is perhaps a good place to comment on "positive" thinking. I often encounter programmers who go on hoping that "something" will solve design and analyses oversights. This sort of sloppy thinking just leads to more and more delays.

Positive thinking works better when positive is interpreted to mean definite, not vague or sloppy. A user who is definite, that is, positive about system requirements is a godsend. Frankly, I run like heck if a user gets the attitude of: "Just make it work."

A program that I wrote some seven odd years ago is still in use. It is a simple little program that totals the money spent on film production at the end of a shoot. Mine was one of three versions, and the only one still in use. The client is a great guy, but when we reached what should have been the end of the project, it seemed to stretch out endlessly. Then I noticed that my client had picked up an encouraging attitude, but had quit giving specific instructions. When I pointed this out, he gave me a stream of very definite requirements and we wrapped up the project in short order.

Rule #9: The most important thing about a system is the most important thing about a system. I took over a project to write a billing system for an accounting type organization. The programmer had developed a wonderful system to track client folders, but

overlooked the fact that the number one priority for the customer was to straighten out confusions in billing.

It is important to prioritize features as this helps when planning delivery schedules and in estimating how much attention each function deserves. It is even more important to discover, announce and verify the *one* thing that will make or break the system. This single item will usually get attention far in excess of all the remaining parts of the system.

For a docket system in a law office it would be to have all the calendar events announced at the right time. If the calendar report is accurate and reliable then the system will be judged as "essentially" reliable, even if the mailing list feature is a disaster.

Summary:

The common denominator to successful systems, one that is uniformly absent from failured projects, is a clear understanding of the system and its contents. A badly written system that limps along losing data will be more useful, if well understood, than one that is smooth and flawless but mysterious to the user.

When I started writing business applications 15 years ago, my intention was to use computers to make organizations more sane. Ironically, the way to accomplish that is to make the computers more comprehensible and take out the mystery or, in other words, to make computers more sane. Computers are tremendously powerful administratively. Misused they can drive a work force into apathy and confusion. Well understood, and with a minimum of care in the design and implementation of the software, they can dramatically grow the very same organization.

Bruce Gilham Senior was the founder and president of DataForce Inc. His involvement with Clarion began in 1989. Bruce was also the founder of the Los Angeles Clarion User Group and sponsor of the successful DevCon West conferences for Clarion developers.

ABC Design Series: The ViewManager Part 1
(Nov 23,1999)

November 1999 News
(Nov 23,1999)

The Art Of Software Development: Analysis By Design
(Nov 23,1999)

Clarion Magazine Version 2!
(Nov 23,1999)

Vol 1, No 10
November 1999

published by CoveComm Inc.

Clarion MAGAZINE

Main Page

Log In
Subscribe

Frequently Asked Questions

Site Index
Links To Other Sites

Downloads
Open Source Project
Issues in PDF Format
Free Software

Advertising

Contact Us

From The Publisher

New Magazine Design To Be Unveiled Dec 7/99

Some of you may have noticed a few minor changes recently in the visual appearance of Clarion Magazine, particularly on the main page. Well, one thing led to another, and as a result in December you'll be seeing a brand new look on the web site.

Why a redesign? When I started Clarion Magazine I decided that no matter how much I liked the initial design, I'd do a review in six to 12 months. The Clarion product constantly evolves, as do the skills of Clarion developers. Clarion Magazine needs to adapt and grow as well.

Of course, it takes time to implement a big change like this, more than is usually available between issues. Clarion Magazine publishes four times per month, on Tuesdays. Four times per year there's a fifth Tuesday, as there is this month, which usually means a week off for the elves that put the magazine together. But not this time. For the next two weeks, if you lean in close to your computer, you may be able to hear the electronic saws whining and the digital paint splashing. Or you might just get a lot of worried looks from your co-workers.

There will also be some changes to the publication schedule as a result of the holiday season. To allow the aforementioned elves a vacation during Christmas (and to celebrate the new look) the December 7 issue will be double in size, followed by regular issues on the 14th and 21st. There will be no issue published on December 28th. You'll still get your usual monthly content in December, and the normal publication schedule will resume on January 4, 2000.

Be sure to come back on December 7th for Clarion Magazine v2.0!

Dave Harms
Publisher

ABC Design Series: The ViewManager Part 1
(Nov 23,1999)

November 1999 News
(Nov 23,1999)

The Art Of Software Development: Analysis By Design
(Nov 23,1999)

Clarion Magazine Version 2!
(Nov 23,1999)