Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

clarion magazine
Good help isn't that hard to find.
$6.25/month

Issue Index

February, 1999

February Issue Index

This issue is also available in PDF format.

on any column, reformat, do quick reports, even export to other apps.
(Feb 15,1999)

David Bayliss On The ErrorClass
David Bayliss begins his comprehensive series on the inner workings of ABC with a discussion of ErrorClass, one of the most fundamental and basic ABC classes.
(Feb 22,1999)

February News
News from the Clarion world (and occasionally places farther afield).
(Feb 22,1999)

The Novice's Corner - Getting A Grip On Clarion
For those of us who have been using Clarion for years, the application development environment, with all of its idiosyncrasies, is second nature. If you've just picked up Clarion, however, your initial reaction may be more one of confusion than familiarity. This article provides a road map to the Clarion development environment, including the use of templates and the "standard Clarion" approach to application design.
(Feb 22,1999)

The Clarion Advisor: Topspeed Driver Error Codes
Nigel Hicks has kindly provided a comprehensive list of TPS driver error codes. The Clarion Advisor has added some redirector information and a survey of which errors were reported in the newsgroups, and when.
(Feb 22,1999)

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

clarion magazine
Good help isn't that hard to find.
$6.25/month

Clarion Magazine Information
November, 1999

The Clarion Magazine Countdown Contest

You can win a year's subscription to Clarion Magazine (value US$75) in our Countdown Contest. The week before the official launch of Clarion Magazine we'll be giving away one subscription every business day. Winners will be drawn at noon, Central Time, on February 1, 2, 3, 4 and 5. If your email address is drawn, you'll have 24 hours from the time we send you your notification email to respond and claim your prize. Winners will be announced the following day.

To enter, just visit the mailing lists page and add your email address to the Notice of Changes list. If you've done this previously, then you're already entered!

Contest Rules: Only one entry per person. Members of Team Topspeed and employees of Topspeed Corporation are not eligible.

Winners!

Monday, February 1, 1999 - **John Claycomb**, APO, AE (Frankfurt, Germany).

Tuesday, February 2, 1999 - **David Pennycuick**, Toowoomba, Queensland, Australia.

Wednesday, February 3, 1999 - No response from the potential winner! Couldn't have been an Australian. A second draw will be held on Thursday.

Thursday, February 4, 1999 - **Jono Gellie**, Hazelbrook NSW, Australia; **Luis De Almeida**, Edenvale, South Africa.

Friday, February 5, 1999 - **John Oerke**, Modesto, California, USA.

Congratulations to the contest winners!

Clarion Magazine Main Page
(Feb 12,1999)

The Clarion Magazine FAQ
(Feb 24,1999)

The Site Index
(Feb 12,1999)

Subscribe to Clarion Magazine!
(Feb 8,1999)

The Subscription Agreement
(Feb 5,1999)

Refund Policy
(Feb 8,1999)

Subscriber Services
(Feb 8,1999)

Information For Authors
(Feb 2,1999)

How To Contact Clarion Magazine
(Feb 2,1999)

The Unofficial Clarion OOP Page
(Feb 6,1999)

Lend Us Your Links
(Feb 7,1999)

Getting Access To Clarion Magazine
(Feb 7,1999)

Clarion Magazine's Automated Mailing Lists
(Jan 31,1999)

Clarion Magazine Is Best Read With Verdana
(Feb 7,1999)

Clarion Links By Category
(May 17,1999)

Free Newsgroups For Clarion User Groups
(May 10,1999)

Free Utilities: Class

Browser And Compile Manager
(Dec 7,1999)

Clarion's Publication Schedule
(Feb 5,1999)

Advertising Policy
(Feb 7,1999)

Open Source Code Available Now
(Mar 8,1999)

The Masthead
(Feb 15,1999)

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

Clarion Magazine Information
November, 1999

What About The Unofficial Clarion OOP Page?

When the London development team first began the move to ABC and OOP there were many objections from Clarion developers who didn't want to be forced to change from procedural programming. As a fairly late convert to OOP I felt I understood some of their concerns. I started the Unofficial Clarion OOP Page (UCOP) as a resource for programmers who wanted to take a stab at object-oriented programming.

UCOP was quite a success, and I've had many comments from people who enjoyed reading it. And it's in part because of this success that I and a number of others began creating a successor to the Clarion OOP Page: Clarion Magazine.

Clarion Magazine is certainly about more than just ABC and OOP, but it seems appropriate to fold some of the material from UCOP into the magazine's electronic pages.

From time to time, articles or excerpts from the UCOP site will appear here. If they're more or less unaltered, they'll be freely available to the public, as they were before. If they've been significantly modified they'll appear as articles in the subscribers-only section.

To all the readers of the Unofficial Clarion OOP Page, my thanks! You helped make Clarion Magazine a reality.

Dave Harms, Publisher

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

Feature Article

February, 1999

From The Publisher

I'm pleased to welcome you to the inaugural issue of Clarion Magazine, an on-line resource for Clarion programmers. Although Clarion Magazine (a.k.a. ClarionMag) is a subscription-based publication, **for the month of February this magazine is completely free.** Be sure to visit the subscription page after you've had a look around – an early bird discount applies if you subscribe before March 1, 1999.

Clarion Magazine is a "live" magazine. Traditionally, Clarion magazines have been monthly publications. Clarion Magazine will be a bit closer to a weekly, and on occasion may be updated several times a week. This is an internet-based publication - why should you have to wait a month between issues?

In terms of content and internal organization, however, you can think of this magazine as a monthly publication. My aim is to meet and exceed the standards of quantity and quality standards set by previous Clarion monthlies.

Why Another Clarion Magazine?

There have been a number of Clarion books and periodicals over the years, but I think it's safe to say that the Clarion community has never suffered from an overabundance of programming information.

Perhaps that's not strictly true if you consider the sheer volume of messages posted on the TS news server (tsnews.clarion.com, if you're not there already) and other places, like comp.lang.clarion and the now almost defunct TopSpeed forum on CompuServe (GO TOPSPEED).

If the problem isn't volume, it is content. How do you find the information you need? How do you educate yourself, and continually develop your own programming skills? How do you stay competitive?

Several years ago I had the opportunity to work with Ross Santos on the book *Developing Clarion for Windows Applications*, published by SAMS. One of our primary goals in writing that book was to give developers the skills they needed to solve their own programming problems. We tried to build, in the reader's mind, a mental model of how Clarion programming works. We wanted to create a road map through the maze of Clarion programming options.

ClarionMag is dedicated to these same goals. Although a weekly magazine format is necessarily different from that of a book (content appears continuously rather than in one fell swoop), the material in this magazine will, I hope, grow into a cohesive whole.

Our Audience

Clarion programming is in the midst of a major shift from procedural programming (with the legacy templates) to object-oriented programming (with the ABC templates). This presents a bit of a challenge for a magazine such as this one. ABC is the future, but there's a lot of code in legacy applications. Clarion Magazine will certainly focus much of its energy on the transition to ABC and OOP, but there will be useful information for legacy coders as well.

There will be an ongoing special series directed at the novice Clarion programmer, as well as some highly technical pieces. For the most part, however, Clarion Magazine will target working Clarion developers who want to maximize their productivity and understanding of the development environment.

Reader Feedback

Clarion Magazine has its own news server for subscribers who wish to discuss articles, get feedback from magazine staff, or collaborate on open source projects. When you subscribe, you'll receive information on how to get access.

The Clarion Open Source Project

Open Source is a concept that is taking the software world by storm, in large part because of the success of its most visible example, the Linux operating system. (Incidentally, this web site is hosted on a Linux box, so ClarionMag is a direct beneficiary of open source software already.)

Clarion Magazine will be hosting the Clarion Open Source Project, a distribution of selected source code for third party products.

Author, Author!

Over the coming months Clarion Magazine will be bringing you articles by some of the best-known authors in the Clarion community, including David Bayliss, Steve Parker, Russ Eggen, Gus Creces, Ross Santos, and Mike Hanson. If you're interested in contributing, please visit our contributor page.

The Book!

I'm very happy to report that Ross Santos and I have recently obtained all publishing rights to *Developing Clarion for Windows Applications*, which has been out of print for some time. Updated book chapters will be finding their way into this magazine in the near future.

What will happen to the Unofficial Clarion OOP Page?

As many of you know, I've been involved in publishing the Unofficial Clarion OOP Page, a resource designed specifically to assist Clarion programmers in making the transition from procedural programming to object-oriented programming. This work now falls under (but by no means completely describes) the mandate of ClarionMag, and as a result the Unofficial Clarion OOP Page has been taken off the 'net. Click here for more.

Enough Talking - On With The Show!

But first, a few words of thanks. Clarion Magazine runs on a Linux server, and **Steve MacLeod** provided essential help getting everything configured and running smoothly, and getting me familiar with my new responsibilities. **Lee White**, of Lodestar Software, did the magazine's graphic design, without which I suspect the budgies and the fish would be the only ones reading the articles.

I'd also like to thank the members of the Clarion Magazine Advisory Board for their invaluable input.

It's been an exciting process bringing ClarionMag to life. I hope you'll get as much satisfaction from reading the magazine as I get from publishing it.

Dave Harms, Publisher

Notes

1. Actually this was Andrew Guidroz's idea. Why do a monthly magazine, when the only reason to do a monthly is because you have to bind paper together and mail it to customers? Why indeed? Back to the article.

Main Menu | Log In | Subscribe | Links | FAQ | Advertising

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

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

Search

Feature Article

February, 1999

ABC or Legacy: Which Templates Should I Use?

by Dave Harms

The introduction of the Application Builder Class (ABC) templates in Clarion 4 radically altered the business of creating applications with Clarion. I believe ABC is as important to Clarion as the first Windows version, or the introduction of client-server database drivers. At first glance, however, it may not be immediately apparent that ABC represents any dramatic shift.

Has the AppGen's feature list expanded dramatically? On the surface it hasn't. There are some nice new features available in the ABC templates (like expanded edit-in-place support, and better browse sorting and filtering, but in its first and second incarnations, ABC mainly recreates the legacy templates (although it does improve significantly on legacy's handling of SQL databases).

At the same time, because embeds are handled somewhat differently, and some template options are organized differently, you will need to relearn a few things. There are costs to ABC as well as benefits, and for most of us the short term cost may (and probably will) exceed the short term benefit. If you have a lot of legacy code, you may find it difficult to migrate some of that code to ABC.

So where's the payoff? Where’s the dramatic change?

Finding The Value in ABC

Software development, as an industry, is forever evolving. We're not using the tools we used five or ten years ago (at least most of us aren't), and chances are not many of us will be using the tools we have now five years into the future. Change is a given, though it doesn't necessarily happen continuously. Often we'll use the same tools, or the same programming approach, for a period of time, until our own needs, or the solutions being provided by our competitors, force us to adopt some new way of development.

Most of us do AppGen-based development, which means we rely largely on whatever TopSpeed's templates provide. (To my knowledge there are no complete template chain replacements commercially available.) The templates determine the feature set and, for many of us, the look and feel of our applications.

The real value in ABC is that while the current template feature set has increased modestly, the potential for what the AppGen can create has increased rather dramatically.

In the legacy templates, the template files themselves contain a mix of template language statements and Clarion language statements. The template language statements control the generation of the Clarion language statements. As you choose options in the AppGen, code is generated or not generated as needed.

While this has been an effective way to create Clarion applications for a number of years, some of the code has become extraordinarily complex and difficult to read. The browse templates, in particular, have been straining at the perimeter of maintainability for some time. Bruce Barrington recently responded to this question in the TS newsgroups, saying the legacy templates "represent the end of a development path. Even the author was afraid to change them for fear of unpredictable side effects."

Many of us purchased the now-defunct PowerBrowse because it added many features that weren’t readily available in the standard browse, but sadly even PowerBrowse became too expensive to maintain over the long haul.

A key problem with the legacy approach to Clarion code generation is readability. Templates often use switches, such as the #IF...#ENDIF structure, to determine when to generate code, as in Listing 1.

Listing 1: Some typical template code.

```
#AT(%BrowseBoxPopupRecords)
  #IF(%Control=%ListControl AND %EditViaPopup)
    #SET(%ValueConstruct,'')
    #IF(%DeleteControl)
      #SET(%ValueConstruct,%DeleteText)
    #ENDIF
    #IF(%ChangeControl)
      #IF(%ValueConstruct)
        #SET(%ValueConstruct,%ChangeText & '|' & ↵
         %ValueConstruct)
      #ELSE
        #SET(%ValueConstruct,%ChangeText)
      #ENDIF
    #ENDIF
    #IF(%InsertControl)
      #IF(%ValueConstruct)
```

```
        #SET(%ValueConstruct,%InsertText & '|' & ↵
         %ValueConstruct)
      #ELSE
        #SET(%ValueConstruct,%InsertText)
      #ENDIF
    #ENDIF
    #IF(%ValueConstruct)
IF %InstancePrefix:PopupText
%InstancePrefix:PopupText = '%ValueConstruct|-|' & ↵
        %InstancePrefix:PopupText
ELSE
  %InstancePrefix:PopupText = '%ValueConstruct'
END
    #ENDIF
  #ENDIF
#ENDAT
```

Note that although the template code statements (those prefixed with #) are indented and the Clarion code statements are indented, the indentations have little if any relationship to each other. It’s often difficult to visualize the generated source simply by looking at the template.

Templates also often contain generated code in temporary variables. The string variable %ValueConstruct is commonly used for this task. Listing 2 shows some code used to create a popup menu string.

Listing 2: Template code to create a popup menu string.

```
#AT(%BrowseBoxPopupRecords)
  #IF(%Control=%ListControl AND %EditViaPopup)
    #SET(%ValueConstruct,'')
    #IF(%DeleteControl)
      #SET(%ValueConstruct,%DeleteText)
    #ENDIF
    #IF(%ChangeControl)
      #IF(%ValueConstruct)
        #SET(%ValueConstruct,%ChangeText & '|' & ↵
                %ValueConstruct)
      #ELSE
        #SET(%ValueConstruct,%ChangeText)
      #ENDIF
    #ENDIF
    #IF(%InsertControl)
      #IF(%ValueConstruct)
        #SET(%ValueConstruct,%InsertText & '|' & ↵
                %ValueConstruct)
      #ELSE
```

```
        #SET(%ValueConstruct,%InsertText)
      #ENDIF
    #ENDIF
    #IF(%ValueConstruct)
IF %InstancePrefix:PopupText
  %InstancePrefix:PopupText = '%ValueConstruct|-|' & ↵
%InstancePrefix:PopupText
ELSE
  %InstancePrefix:PopupText = '%ValueConstruct'
END
    #ENDIF
  #ENDIF
#ENDAT
```

The actual generated code for this block may look like listing 3:

Listing 3: Some template-generated code.

```
IF BRW1::PopupText
  BRW1::PopupText = '&Insert|~&Change|~&Delete|-|' & ↵
        BRW1::PopupText
ELSE
  BRW1::PopupText = '&Insert|~&Change|~&Delete'
END
```

Templates, like programs, often need to reuse the same code. In a template, the #GROUP statement (not to be confused with the Clarion language GROUP statement) defines a block of template code which can be used much the way a procedure is used in an application, except that the job of the #GROUP, like the rest of the templates, is really to generate Clarion code. #GROUPs can even take parameters, like procedures.

Listing 4 shows a #GROUP (from STANDARD.TPW) that formats a filter.

Listing 4: A #GROUP that formats a filter.

```
#GROUP(%StandardWriteViewFilter,%ConstructedFilter)
#IF(%ConstructedFilter)
%ListView{Prop:Filter} = |
  #SET(%ValueConstruct,%ConstructedFilter)
  #SET(%HoldConstruct,'')
  #LOOP
    #IF(LEN(%ValueConstruct) > 70)
      #SET(%HoldConstruct,SUB(%ValueConstruct,71, ↵
LEN(%ValueConstruct)-70))
      #SET(%ValueConstruct,SUB(%ValueConstruct,1,70))
    #ENDIF
    #IF(%HoldConstruct)
'%'ValueConstruct' & |
      #SET(%ValueConstruct,%HoldConstruct)
      #SET(%HoldConstruct,'')
```

```
   #ELSE
'%'ValueConstruct'
     #BREAK
   #ENDIF
  #ENDLOOP
#ELSE
%ListView{Prop:Filter} = ''
#ENDIF
```

Any place you wish to use this code, you reference the group as follows (in this case, the #GROUP is used with a file drop control):

```
#INSERT(%StandardWriteViewFilter,%DropViewFilter)
```

In order to read and understand templates, then, you will need to be adept at differentiating between template lines and generated code lines; you should keep in mind the expected contents of calculated string variables; and you should know, or be able to determine, what code any given #GROUP is capable of generating. This is, to say the least, a daunting task.

Additionally, there are no visual development tools available for the template language, except for the limited default window and report editing which can be done in the template registry.

There's no doubt that templates have made for tremendous productivity gains. The problem is that the template language, in the legacy templates, was increasingly being asked to do a job it isn't ideally suited to do.

The ABC templates address the weaknesses of the legacy templates by putting the template language back in its proper place, and relying on a class library for the tricky Clarion source code.

Every Template In Its Place

If the templates aren't there to generate complicated Clarion source, then what's their purpose? Ideally, to give the AppGen intelligence. The AppGen isn't so much a program as an interpreter of templates, in the same way that the old DOS BASIC program is an interpreter of BASIC language statements. The vast majority of what the AppGen does, it does under the control of the templates.

A good portion of the templates is made up of statements like #PROMPT, #TAB, #BUTTON, and others, all of which present information to the user and/or accept input. There are a few built-in features, such as the procedure properties form and the global prompts, but by and large what you see in the AppGen isn't the AppGen itself, but the result of the templates you're using.

The real role of the templates is to translate the visual AppGen interface into your final code and to do so in the most efficient way possible.

A Division Of Labour

One of the reasons ABC is so important is because it splits the responsibility for creating your application between the ABC templates and the ABC class library proper. Template code isn't the most efficient way to write programs, for the reasons already explained. That's better done with Clarion source code. Ideally, you'd write as much of your browse code or your form code in Clarion only, and then have the templates create such calls as were needed into that library of code. In this way, the core functionality of, say, a browse procedure, is all Clarion code and much easier to debug and maintain.

That’s not to say you can’t write a generic browse with procedural code. In fact, the predecessor to PowerBrowse, a product called Power Windows, did just this for Clarion 2.x. One drawback to using Power Windows was that you needed to pass up to 40 parameters to the generic browse procedure! This was a complex enough task that the product came with a program that asked you to fill in the fields on a form and generated the procedure call based on your responses.

Perhaps the greatest difficulty in using a procedure library to perform generic, yet complex, processing, is that you have only one point of entry into a given procedure. If your generic browse is one procedure, you have to pass it all of the information it needs to work (like PowerBrowse) or you have to rely on global variables, which is generally not a good idea.

If you aren't going to use a single procedure, then you have to break up the logic into numerous procedures, and you may have to determine at run time which of these sub-procedures you’re going to call. Now you’re probably looking at using procedure pointers. Why not just get out a C compiler and shoot yourself in the foot properly?

Happily, object-oriented code is much better suited to this kind of task than procedural code. The core unit of object-oriented programming is the class, which is made up of blocks of code called methods (which are analogous to procedures), and data, which the methods share. It’s possible to build a class which models quite complex behaviour, and even to combine several classes into a functioning unit.

In the ABC templates, most of the core functionality of your application is contained in the Application Builder Classes, for which the ABC template chain is named. As a result, the templates no longer have to generate the same kind of complex code that the legacy templates generate.

A Better Role For The Templates

In ABC, the role of the templates has changed significantly. Instead of creating the majority of the source code for your application, the templates manage the interface to the ABC class library.

Class libraries are like other code libraries – they’re collections of commonly-used code. In the case of ABC, the library contains code to open and display windows, to load browses with data, to manage scrolling, to update records, and so forth. But ABC, because it’s totally generic, has no specific knowledge of your application’s data, or what you might want to do with your application.

Something like a browse procedure now takes a lot less code than it did with the legacy templates because most of the functionality exists in the class library. In order to use that library, however, you have to know which class methods to call.

That’s where the templates come in. After you choose (in the AppGen) what options or properties you want your browse, or form, or other window or control to have, the templates create the necessary calls into ABC. There is still some generated code, but it’s a lot less than it used to be. More importantly, the core logic is in self-contained, generic classes which can be tested and debugged independently of the templates.

You could, if you wished, create quite large applications entirely with hand code and the ABC library, omitting the templates and the AppGen entirely. It would be work, but a lot less work than creating the same application without ABC.

Additionally, more and more of us seem to feel the need to create larger, more complex procedures, particularly those populated with browse boxes. This isn’t always a problem with 32 bit code, but if you’re writing 16 bit apps you’ve probably bumped up against the 64k segment size limitation. Because much of the code in an ABC app is in the class library, the generated code, and therefore the procedure size, is much smaller than in legacy apps.

Making Your Choice

In the long run, ABC is the logical choice. Unless some white knight takes over the legacy templates, there will be no significant development done in that area anymore. And even if someone does take on this task, it will be a maintenance effort at best. You won't see many ABC-type features being added to the legacy templates, because they're just too difficult to implement.

Switching to ABC does require new skills, but you don't necessarily have to be an OOP expert to become productive. Your required level of OOP knowledge will to some extent parallel your current procedural programming expertise. If you write lots and lots of embed code, you're obviously familiar with the Clarion language. To get the most out of your embed code in ABC, you'll need to know something about the ABC library and OOP in general. If you really just use the AppGen and the templates, and don't write much of your own code, then not a lot is going to change for you.

Whatever your current level of proficiency, you achieved it at some cost. You had to invest in your own skill set. ABC represents a significant technological step forward, and as such it demands a new investment in software development skills.

If you have the option, starting out with a small ABC project is preferable to overhauling a large legacy code base. Unless you're already fully conversant with OOP and have a good grasp of the ABC library, there's a good chance you'll want to rethink a large system redesign once or twice anyway. ABC offers a number of interesting opportunities for large system redesign, some of which I'll be addressing in later articles.

Where Is ABC Headed?

The move to ABC has eased the pressure on the template language by moving most of the core code into the class library. That code is also more extensible than the legacy code. The key principle of object-oriented programming is reusability, and OO languages have a number of mechanism available to make it easier to add new features to a class library without breaking old functionality.

Although ABC in its first and second incarnations mainly duplicates the functionality of the legacy templates, it does provide some new features (such as edit-in-place and query-by-example, in C5), and more importantly lays the groundwork for changes to come.

I expect ABC’s impact to parallel that of the Clarion for Windows 1.0 templates, particularly in the third party market. Those templates were the first that allowed users and vendors to easily plug their templates into Clarion’s templates. It took a year or two before the full import of this was realized, but as any glance at the Accessories catalog will tell you, the vast majority of third party products are sold as templates.

In the same way, as users and vendors realize the power of ABC, new products emerge, and old products are released with new capabilities. Templates themselves won't go away, because templates are still the ideal way to manage classes in the AppGen environment. The impact of the move to OOP, however, will be at least as big as it was for CW templates.

David Harms is an independent software developer and the co-author with Ross Santos of Developing Clarion for Windows Applications, *published by SAMS (1995). He is also the editor and publisher of Clarion Magazine.*

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

Feature Article

February, 1999

The Clarion Open Source Project

Important Notice

Since this article was written CoveComm Inc., publisher of Clarion Magazine, has created the Developers Open Source Public License for use in Clarion-related open source projects. The LGPL referenced in this article is no longer recommended. All Clarion Open Source Project pages are now also available to the public.

You've probably heard the term "open source" bandied about, and you might be unsure as to its meaning, much less how it applies to Clarion programming.

Open source is a way of distributing software that goes back to the GNU* project, although the GNU folk like to use the term "free software" rather than open source. GNU stands for GNU's Not Unix, which is the sort of self-referential acronym you'd probably expect a bunch of programmers to come up with. That's the acronym; GNU itself is (or is intended to be) a "complete Unix-compatible software system."**

What, you say? Sounds like Linux? You're not far wrong. Although Linux is a separate effort in most respects, it has been developed under the GNU General Public License, which is one of the keys to the open source/free software movement. Linux is an amazing success story in that it's a rather complex yet stable piece of software created through what would seem to be an completely chaotic development model. Although there is an organization, and a driving force (Linus Torvalds), no one is really in control of Linux, in the usual way that a person or corporation typically owns and controls a piece of software.

Copyleft

Linux is a collaborative work. Developers all over the world contribute code, and modify other developers' code. Linux, like other GPL'd software, is copylefted, rather than copyrighted.

Copyrighting means that the author of the work reserves all rights to that work, so that no one can change the work without the author's permission. Copylefting means that the author gives anyone else the freedom to use, modify, or distribute the work, but only if they agree to keep the distribution terms unchanged, which means that their changes must be freely distributable also.

(Among other things, this means that Microsoft can't buy Linux. It isn't for sale. And Microsoft, or anyone else, can improve Linux and offer it for sale, but they also have to provide the source under the same terms as the original source.)

GNU and Clarion

Although GNU is a Unix-compatible software system, the GNU GPL can be applied to software developed for any platform. For software developers, the GNU license can be an effective way to exchange code with other developers to mutual benefit. Most Clarion programmers benefit from some sort of informal code exchange already. For instance, there are any number of free templates and source code examples available to Clarion programmers from a variety of sources. Many of these carry some sort of copyright notice. Few are covered by the GNU license.

One of the problems with the GNU GPL is that it was written with entire programs in mind, not parts of programs, which is what most Clarion developers are likely to make available. Source code libraries are, however, covered under the GNU Library General Public License, which has provisions appropriate for licensing code that will become part of a larger work, which may or may not be covered under a GNU license.

The Clarion Open Source Project

Clarion Magazine is initiating the Clarion Open Source Project (COSP) to encourage Clarion developers to put their freeware libraries under the GNU LGPL. Specifically, subscribers will have access to the following services:

- Newsgroups for discussion of COSP libraries.
- On-line documentation for COSP libraries.
- Some level of task listing for programmers wishing to participate. This includes bugs to be fixed/features to be added/documentation to be written.

We believe the LGPL is an excellent means of fostering the development of quality free third-party products. Although anyone can place software under the LGPL, Clarion Magazine reserves the right to choose which products are included in the COSP distribution. You'll know from the on-line documentation the current status of any given product, and you'll be notified of upgrades.

Everyone, not just subscribers, will be able to download the COSP libraries. That's in keeping with the LGPL, which states that any source placed under the license must be freely available.

To kick things off, CoveComm Inc. has released two classes, cciDebugClass and cciProfilerClass, under the GNU LGPL. At present there is almost no documentation included with these classes. Their functionality is as follows:

cciDebugClass

- basic trace message capability (a method call to add a trace message to a log)
- view the trace log in a window
- view the trace log in a toolbox (so you can see messages as they are logged). Switch between tree and list view.
- write the trace log to a file

cciProfilerClass

- derived from cciDebugClass
- uses the profiler hooks to automatically generate a procedure call tree (using tree view by default, but can switch to indented-text view)

You may also wish to view these classes to see how to put your own code under the GNU LGPL.

Download CoveComm's debugging and profiling classes

Both classes will be featured in articles in Clarion Magazine; cciDebugClass is currently scheduled for the middle of March.

COSP and Third Party Vendors

The Clarion Open Source Project is not intended to in any way be direct competition to vendors of third party products. Its purpose is to organize and improve the kinds of contributions Clarion developers have always made to the programming community.

If anything, third party vendors can benefit from COSP. Since the COSP distribution is freely available, anyone who wishes to can provide paid support for the products included in the COSP distribution (although you should keep in mind that the documentation in Clarion Magazine about COSP products is copyrighted and therefore not available for resale by anyone else).

For More Information

Further information about COSP will be posted during the month of March, including product documentation, bug lists, and work to be done. If you're interested in placing some of your code under the GNU LGPL for inclusion in COSP, send an email to cosp@clarionmag.com.

Endnotes

* The G in GNU is pronounced, usually as a hard G. If you didn't pronounce the G,

GNU would sound like 'new' and this would cause a lot of confusion as you tried to explain what you meant by the 'new' license or your 'new' compiler.

** GNU has been around for a while - the original GNU manifesto dates to 1983.

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

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

Search

The Clarion Advisor

February, 1999

Speed Up Your Debugging With A PRJ

If you get an error when compiling an application, and the offending code is in an embed point, the IDE will (usually) take you to that embed point so you can correct the problem. If the error is in the generated code, or in an included source file (such as class source), you'll be taken to the source editor.

The only problem with this is that when you're fixing errors, the editor window is modal. You can't switch to or load another file. This is only a problem when you're using applications, however. If you're hand coding, you get a normal MDI source window on a compile error. Therefore the way to get around the modal error editor window is to temporarily treat your application as if it's a hand coded program, and to do this you need to create a project file.

In order for the compiler and linker to create an EXE or DLL from Clarion source, they need a set of project statements. A typical set of these statements is shown in Listing 1.

Applications have the project statements embedded in the .APP file, but it's quite easy to get the information out of the app and into a project file.

First, export the application to an *appname*.TXA file, using the File|Export Text menu option. Then search *appname*.TXA for this string:

```
[PROJECT]
```

Copy everything from the line following [PROJECT] to the start of the next header, which should be [PROGRAM]. The last line in the project is normally the #link statement. Paste this source into a new file, and give it the name appname.PR or appname.PRJ.

There's no difference between a PR file and a PRJ file except that if you open a PR file with Clarion it will open as a straight text file, and if you open a PRJ file Clarion brings

The Clarion Magazine Countdown Contest
(Feb 1,1999)

The Unofficial Clarion OOP Page
(Feb 6,1999)

From The Publisher
(Feb 8,1999)

Feature: ABC or Legacy
(Feb 8,1999)

Feature: The Clarion Open Source Project
(Feb 8,1999)

The Clarion Advisor: Speed up your APP debugging with a PRJ
(Feb 8,1999)

Interview: Roy Rafalco (Part 1)
(Feb 15,1999)

Feature: Don't Know, Do Care - A Philosopher Looks At OOP
(Feb 15,1999)

Product Review: Xplore Templates
(Feb 15,1999)

David Bayliss On The ErrorClass
(Feb 22,1999)

February News
(Feb 22,1999)

The Novice's Corner - Getting A Grip On Clarion
(Feb 22,1999)

The Clarion Advisor: Topspeed Driver Error Codes
(Feb 22,1999)

up the project editor, which is what you're used to seeing when you choose Project|Edit while you have your application open.

Close your application and use Project|Set to choose the .PR or .PRJ file you created. Now compile your application. When the compiler comes to an error, it will display it in a non-modal editor window, and you can load whatever other source files you like.

This approach is also helpful when you have a really nasty bug to fix. Sometimes the only way to do this is to cut away at your program until you've reduced it to the smallest possible program that duplicates the error. Rather than go through a bunch of change/generate/compile cycles, create a text project and hack away at the source to your heart's content. You only need to regenerate the app to get back to your original code.

Listing 1: Project statements for compiling a small 32 bit application with full debug.

```
#noedit
#system win32
#model clarion dll
#pragma debug(vid=>full)
#pragma define(profile=>on)
#compile "cciprof.clw" /define(profile=>off)
#compile "ccidebug.clw" /define(profile=>off)
#compile "TESTBC0.CLW" /define(GENERATED=>on) -- GENERATED
#compile "TESTBC.CLW" /define(GENERATED=>on) -- GENERATED
#compile "test.clw" /define(GENERATED=>on) -- GENERATED
#compile "test001.clw" /define(GENERATED=>on) -- GENERATED
#compile "test002.clw" /define(GENERATED=>on) -- GENERATED
#compile "test003.clw" /define(GENERATED=>on) -- GENERATED
#compile "test004.clw" /define(GENERATED=>on) -- GENERATED
#compile "test005.clw" /define(GENERATED=>on) -- GENERATED
#compile "test006.clw" /define(GENERATED=>on) -- GENERATED
#compile "test007.clw" /define(GENERATED=>on) -- GENERATED
#compile "test008.clw" /define(GENERATED=>on) -- GENERATED
#pragma link("C5ASC%X%%L%.LIB")
#pragma link("C5MSS%X%%L%.LIB") -- GENERATED
#pragma link("C5ODB%X%%L%.LIB") -- GENERATED
#link "test.EXE"
```

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

Feature Article

February, 1999

Interview: Roy Rafalco

A Conversation With Topspeed Corporation's New CEO

On December 28, 1998, Topspeed announced the appointment of Roy Rafalco, Topspeed's President and Chief Operating Officer since 1993, to the position of Chief Executive Officer. Rafalco succeeds Bruce Barrington, Topspeed's founder, who will remain Chairman of the Board. Click here for the full press release.

Clarion Magazine: Congratulations on your appointment as Topspeed's new CEO.

Roy Rafalco: Thank you. It's a thrill for me to be put into this role, one that I did not seek. It's hard to fill the shoes of Bruce Barrington. Although he's not going anywhere. He'll be just as active as he always has.

As a CPA you've had a wide variety of business experience before coming to Topspeed.

I've been in public accounting because of the CPA background. I've been in city government, consumer electronics, and in fabrics and wallcoverings.

But you're also a lawyer by training. Why haven't you gone into law?

I've never wanted to be in a law firm, but I do enjoy business, I like to develop strategies, and work with people. But I'm a masochist. I didn't have to get a law degree. (Don't tell my wife.) After being out of school for a couple of years, I got bored. I always had this goal of getting a law degree. I didn't think it would hurt too much to add that to my CPA background.

You've had some international experience as well.

At the time I was with RCA/General Electric, and I was flying to Taiwan regularly. It was a good experience. The reason I was in all these industries was to round out my background.

Where were you before you came to TopSpeed?

With Payne Fabrics. It was good experience. I went out on the road with fabric sales people, I went to design centers, I dealt with a much different clientele, ones who were interior designers. So you can imagine that kind of a customer base versus the kind of customer base I have today. A much different mindset. There you learn about image, you learn about things that are different than technicalities. It’s all color, design, and style. It’s all about the moment, and how you feel about things.

After Payne Fabrics you started looking for something different. That was 1991?

It was a recessionary period then, and after a lack of success using search firms, I decided I’d do my own direct marketing campaign. I did a thousand-piece mailer to companies all over the country. I decided that I want to target the growth industries, so I picked pharmaceuticals, health care, computer hardware and software. I set a minimum for annual sales revenues. As I was reviewing my list, I found this Clarion Software Corporation located in Pompano Beach Florida. It was below my minimum sales revenue threshold, but I said to myself I’m going to add that to my list. I manually overrode my search criteria.

Each letter went to all the company’s CEOs, so Bruce Barrington received the letter. Later I received a call from the president of the company at that time. Before I came down I looked up Clarion, and golly, look at this, PC Magazine editor’s choice awards, wow, these guys have the technology. This sounds great!

When I came down, the president interviewed me. He said "You’re coming in as CFO because our financial situation is in a mess. This company may not be around after a year. But we’ve got the technology, we can turn this thing around."

And then I sat down with Bruce. Bruce didn’t take a whole lot of time with me, but he came across very wonderful, warm and caring. He talked about how he saw the vision for the company and that the JPI merger was something that was really critical to our success. He liked my legal and CPA background because of the merger and all the things that had to be done. Also Bruce has always been interested in taking the company public, and with my background he could see where I could help position the company to maximize value.

I went back to my family in Dayton, Ohio in January with about a foot of snow on the ground. They heard about Pompano Beach, just north of Fort Lauderdale. I said to my wife you know, this company may not be around in a year. And my wife just totally blew it away. Nope, Fort Lauderdale. What’s the temperature down there? Do you think we can get a pool? I had no choice. The family had made up its mind, the kids had made up their mind. They wanted to go to Florida.

It was a big risk.

That’s right. I’m such a conservative guy. You may have seen me in some pinstriped suits. For me to take this kind of a position was really uncharacteristic of my judgement. But I said to myself it’s high tech, high return, it also has to be high risk. So if it doesn’t work out. I’d rather be walking the beach in Fort Lauderdale than the streets of Dayton Ohio.

On Feb 24, 1992, I came on board and started with the company. The merger hasn't even been clearly defined yet, and it's got to get closed somehow. The acquiring company is losing money. We have none of our upgrade revenue, and when is CDD 3 going to come out?

And you're thinking Ohio is looking pretty good right now.

It's like holy cow, what did they throw me into here? I have to say that Bruce was a great help and support. It was really hairy because on my 60th day with the company the president was terminated. The first thing I'm thinking of is this is the guy who brought me in, and my God, I'm on the way out. But Bruce said "I don't want you to worry about anything, I like you, I like your background, I think you can really help us". And at that point Bruce told me I would continue in my role. He did tell me he wanted me to take over operations in addition to the financial side. He was going to take on the president's role in addition to the CEO role.

And then that was the same year Data General sued us.

What did they sue over?

Clariion – Clariion with two "i"s. There was a whole Wall Street Journal ad advertising Clariion disk arrays. There was a big trumpet and all this kind of stuff. I was out of town, and Bruce went to our outside legal counsel and said we need to do a demand letter, so we sent off a demand letter and within a few days they'd sued us. They seized the legal jurisdiction in Boston, where they're headquartered and where there is a more favorable interpretation of trademark laws.

Those were dark days. How long before things started to turn around?

It was several years. We were in a pit. I guess we started seeing light at the end of the tunnel, it wasn't until late 96, 97. That's a lot of years! But you know, through those years I believed in Bruce's vision of how the London developers were going to help us. In addition, I have Clarion under my skin. It's hard to describe, but it's the belief that we can free the programmers from writing so much code. And then of course, the customer base put their arms around me, and how can I go anywhere else? How can I have any friendlier customers? Where can you find customers who are loyal and really pulling for you? And I can tell you there were some really tough times at the company. There were some key customers that I leaned on. They said Roy, we believe in you, we know it's tough, we know you'll make it, if there's anything we can do for you, give us a call. And you can't find too many customer bases like that.

Do you think Clarion developers feel so strongly about the product because it's such a small community?

That's part of it but you know what, it's a belief that programming does not have to be so difficult. I personally have seen the benefits of the tool. And I make it a point to talk to customers about the benefits of the tool, about all the good things and all the bad things. It's such a shame that programmers as a whole who don't use Clarion have to deal with the things they have to deal with.

We're out to convert the masses to something that can make their lives a whole lot easier. We're out there to save the programmer from all of this work and elevate them to a level of business analyst and manager, rather than a coder.

How do you think the Clarion programming community has changed over the last few years?

I still sense that most of them are independent software developers, and that really hasn’t changed. I mean we have tried to break through into other areas, and in some cases we’ve been successful. Well, why are our customers independent software developers? I believe there is a common personality trait among all our customers. And what is that profile? One is time is their most important asset. Two, they’re mavericks. Three, they’re risk takers. They don’t necessarily follow the rest of the world. And they certainly don’t think MS is the technology leader. And the list just goes on and on and on.

Another typical thing is that Clarion programmers end up being very successful, whether it’s monetarily, or leisure time, or just love of using the tool. Their quality of life is enhanced. The biggest kick I get is getting calls on the phone where I want to thank TS for "buying my new house for me," or "my new car." That really charges me up.

Now the frustrating thing about the Clarion developer, and I don’t know how to get around this, is that same person who’s very happy, thanking me over and over again, and saying "If there’s anything I can do for you, you tell me," won’t do what I ask. I say to them "You know, there’s just a very simple thing I need for you to do. Do you have a local general PC user’s group? "Oh yeah," the customer replies. "Do you attend it?" I ask. "Oh yeah," the customer replies." "Well what I want you to do is to set up one of those meetings for you to highlight Clarion," I ask. "Oh no, no, I’m sorry, I can’t do that," the customer replies." "Why?" I ask" "Well that’s my competitive edge," replies the customer.

It hasn’t happened just once. It’s happened to me over and over and over again. Of course, I try to explain to the developer that I understand. But, listen, if you want us to grow, we have to grow our masses. And you’ve got to talk about us. You’ll still have the edge. You have the experience. We really need your help to get the word out.

Let me take the other side. We’re the developers, you’re the software company. You’re supposed to be marketing it.

I understand and agree with that viewpoint.. Part of our problem has always been that we’re in competition with MS. And how do we price our tools? Do we increase the price in order to fund more marketing? We’re in a catch-22 because of resources. We have tried to get the word out, and based on the brief history of the company I've given you, you can understand that we were under no circumstances able to do much of anything but consolidate, lick our wounds, and bring the technology along.

A lot of companies would have folded in those circumstances.

Absolutely. The only reason we’re here is due to the tremendous support of the shareholders, and the sheer will of the management and employees who just said we’re not going to let this die, we’re not going to let this happen. We’re going to scrap, we’re going to do whatever we can to make it happen, and believe me, I could share some very innovative ways how we got through this morass. Of course, the ultimate support came from our customers who have been very patience and loyal over these many years. To this day, Bruce and all of us are amazed how we were able to pull through those times.

Next month in Clarion Magazine: Part Two: Roy on Linux, marketing, and the future of Clarion.

Clarion MAGAZINE
Clarion Development Resources
published by CoveComm Inc.

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

Search

Feature Article
February, 1999

Don’t Know, Do Care

A Philosopher Looks at OOP

by S.H. Parker

It has been my privilege and my pleasure to write about Clarion for six years. While often taxing it has earned me the sobriquet "the guy who writes Clarion for the rest of us." I wear that with great pride.

By training, however, I am a Philosopher. The creation of a new Clarion publication seems an appropriate time to "come out of the closet," wedding my vocations.

How We Got Here

Some years ago, the war cry at the Microsoft Developer’s Conference was "Windows, Windows, Windows." Topspeed’s rejoinder at Devcon ‘96 was "Templates, Templates, Templates." At Devcon ‘97, David A. Bayliss, in his inimitable manner, brought forth Topspeed’s latest war cry, "Don’t know, don’t care." Because the objects in C4 handled [whatever it was that required handling], he claimed, we didn’t need to concern ourselves with what was happening inside the gray box of the object (since we are supplied with the code for the classes, it really isn’t quite right to call them "black boxes").

While DAB was probably overstating for the sake of making his point, most of us do care what is happening inside those objects, especially when they don’t do what we want them to. It is not only a new way of thinking and doing for many of us, Topspeed’s implementation is different from what we may have been expecting based on our reading.

The Clarion Magazine Countdown Contest
(Feb 1,1999)

The Unofficial Clarion OOP Page
(Feb 6,1999)

From The Publisher
(Feb 8,1999)

Feature: ABC or Legacy
(Feb 8,1999)

Feature: The Clarion Open Source Project
(Feb 8,1999)

The Clarion Advisor: Speed up your APP debugging with a PRJ
(Feb 8,1999)

Interview: Roy Rafalco (Part 1)
(Feb 15,1999)

Feature: Don't Know, Do Care - A Philosopher Looks At OOP
(Feb 15,1999)

Product Review: Xplore Templates
(Feb 15,1999)

David Bayliss On The ErrorClass
(Feb 22,1999)

February News
(Feb 22,1999)

The Novice's Corner - Getting A Grip On Clarion
(Feb 22,1999)

The Clarion Advisor: Topspeed Driver Error Codes
(Feb 22,1999)

The Received View

The almost insurmountable intellectual problem I've always had with OOP is its essential arbitrariness.

Consider this introduction to "inheritance" (from Gary Entsminger's well-received The Tao of Objects):

> Once you've defined a new base type (or class), you can build on it using inheritance.... For example, if vehicle is a base class and you derive a car class from vehicle, car inherits some basic properties and behaviors from the abstract vehicle class. But there's more to a car than that....
>
> Let's represent this system with a base object type (or class) called vehicle and derive objects called bicycle, car and boat. Any vehicle can be steered, and vehicles do different things to steer themselves ... Vehicle can respond to a message called steer, so any type that inherits vehicle can also accept that message.

Later, he uses the most apt analogy:

> Another way of expressing OOP: You build family trees (or hierarchies) for data structures.... each subclass has a single immediate ancestor.

And, quite to the point, each hierarchy has a single root (root system).

Not to pick on Mr. Entsminger, his treatment is entirely typical, but "Why stop at 'vehicle'? After all, it came from somewhere; it has a 'root'." Similarly, "Why start at 'vehicle'? My app is about boats; why not start with 'boat'?" (Human beings have a unique ability to comfortably entertain entirely contradictory beliefs.)

Once you start down the slippery-slope of inheritance, there simply is no logical terminus (until you hit the fundamental building block of nature, whatever it is). If this puts you in mind of the theoretical problems that atomic theory faced in the early parts of this century and appears to be facing again, you're not far from the mark.

Yet, this is fundamental to OOP. It is the logical consequence of "inheritance." Mr. Entsminger, himself, hints at the problem: "Once you've defined." Here "you've defined" means "you have decided." And this is essentially arbitrary.

It also leads to the other side of the issue, "Why start at 'vehicle'?" Recognizing the underlying *post hoc* nature of the exercise (recent research indicates that "reason" simply ratifies what has already been "decided"), if your application is for boat sales, why not simply decide that "boat" is your base class? Why not indeed.

The problem with modeling real-world objects is "What is the 'root' of the hierarchy? How do you know when you've hit the primitive components?" In Philosophy, this issue was determining the essence of an object or group. What, for example, is essential to being "human?"

Plato addressed this issue. He saw all of perceived reality as combinations of 12 "Forms." By the Middle Ages, European scholars were down to four elemental parent classes: earth, fire, air and water. But by the late eighteen century, Kant had us back to 10 or 12 "Categories."

"Inheritance" and "composition," obviously, are not exactly new concepts.

Only the Bible is logically complete in addressing the underlying issue of derivation: everything is derived from God. The tetragrammaton (the four letter name of God), on the standard reading, means "I am, I will be." I.e., "The everlasting." By definition, there is no appeal beyond that which is eternal. Even if you are a non-believer, you must admit, this is the only model which satisfies the hierarchical requirement of our thinking about OOP.

So, it is no wonder that OOP was slower off the mark than journalists expected. Few, if any, took the time to analyze this, though many programmers were uncomfortable without quite knowing why (and now *you do*). All kinds of objects were spoken of, but "files" were never mentioned. And just how does one write business programs without files? What does the universal ("universal" they had to be since objects were to be off-the-shelf components) "student" object look like?

Too hard for you? Ok, what are the properties and methods of the (universal) "person" object? Bertram Russell answered this one. A developer of modern set theory, Russell was able to demonstrate that the set of "featherless bipeds" uniquely described humankind. While not immediately apparent to many, the absurdity of Russell’s demonstration crushed the entire hierarchical approach to "essential properties" (which is also implicit in presentations of OOP) until resurrected by Chomsky and Kripke in the 60’s.

The Topspeed View

The arbitrary nature of OOP is actually the core problem in Philosophy of Language.

In the 1910’s, philosophers began to look at language in earnest. "How do **words** ‘mean’?" they asked, "How do words reflect the essence of things?"

By the 40’s, the "ordinary language analysis" school of Philosophy had started to claim that we all understood perfectly well what words meant. Therefore, the correct question is "What is the meaning of ‘meaning’?", i.e. "**How** do words ‘mean’?" "Essence" was side-stepped. (And the issue wasn’t resolved until J.L. Austin did so in the early 60’s.)

The answer lies, in part, in understanding that the denotation, the literal meaning, of words is indeed arbitrary but that the actual meaning of a word or phrase is context-dependent.

It is, I think, no small accident that the developers of Topspeed’s compilers and templates are graduates of the same university where many of the great ordinary language Philosophers practiced.

It seems to me that what they’ve done is said "Look, we all know perfectly well what an ‘object’ is in the context of a (Windows) program we are writing" (this is the "art" constituent of programming).

Now, since Topspeed writes programs that assist in program writing, that context is further refined. So, they did not create objects in the sense that our reading on OOP would have lead us to expect.

Instead, they recognized that a program is a thing (i.e., an object), an object with components. Many of *those components* appear in different kinds of places and many can be extended in useful ways. Those components can be modeled and those are the classes Topspeed created.

Topspeed has done something literally revolutionary in the world of OOP. There is no recognizable *thing* from the external world, in the expected sense, modeled in the Clarion classes. What then are the classes provided? Simply: the classes we use when we use Clarion OOP are actually the building blocks of our programs.

> We create windows. Thus we have a window object.
>
> We create browses. Thus we have a browse object.
>
> We (batch) process files. Thus we have a process object.
>
> We work with files. Thus we have a file object.

(The file itself is not an OOP object but its handling *is*.)

To repeat: a program is an "object" which can be modeled in classes. The ABC classes are, by and large, just the components of a Windows program. While most proponents of OOP are busy trying to figure out how to create functional programs out of non-existent classes describing "real-world" objects, the Development Centre was busy figuring out the building blocks of a Clarion program.

Clarion OOP is just a different way of doing the same things we've been doing all along.

"A rose by any other name"

Because the most used components of a program are encapsulated in classes, you will notice some differences in how you need to code. Specifically, much of what used to be in a single long code listing is now in multiple procedure calls, you will have scoping issues and you'd best be comfortable with parameter passing; these are well documented as are the more commonly used embeds.

"'Some' differences?" you say, "Are you saying that nothing really has changed?" Substantially, this is just what I am saying and, substantially, it is true. `Access:Customers.Open()` does just what

```
IF Customers::Used = 0
  CheckOpen(Customers)
END
Customers::Used += 1
```

does in the Clarion templates. That is, most of the method calls you make just call procedural code in the class library (a CPU still only works one way, after all).

The big advantage is that because of this encapsulation, it is possible to extend the capabilities of the templates in ways that the procedural templates could not be extended (but that's a subject for someone who knows what they are talking about). But an obvious example is `Relate:Customers.Open()` which opens not only the Customers file, but all related files in a single statement.

The point, the important point, is that Clarion OOP, because Clarion is a computer programming language, takes a program as its object. If you were expecting vehicles or employees, you were looking the wrong way (you need to create *those*).

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.

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

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

Search

Feature Article

February, 1999

Xplore Templates

Template add-on lets users customize browses, generate quick reports, export data.

Topspeed's move from the legacy templates to ABC/OOP has brought challenges and opportunities to third party developers. The challenge is to port existing products to OOP/ABC; the opportunity is to make use of and derive from ABC classes. One of the first products to take advantage of ABC was Datamatrix's Xplore, a browse add-on that allows end-users to greatly modify the appearance and behaviour of ABC browses.

Xplore is the creation of Brian Staff, who initially wrote the product to add the ability to sort browses by clicking on headers, much the way Windows Explorer lets you sort file lists (hence the name Xplore). The good news for Clarion developers is that Xplore has been hit by some serious feature creep since then, and the name Xplore really doesn't do justice to all of the capabilities of this product. Xplore lets your users customize the appearance of their browses in almost any way imaginable, and provides useful data export and quick reporting/graphing capabilities.

Xplore was originally available for C2 legacy templates, and the three products that made up the original Xplore suite (Xplore ReFormat, Xplore ReOrder, and Xplore Column Properties) are still available from Datamatrix, though they are no longer being enhanced. The current product is the ABC version (for C4 and C5), which includes all of the features of the original suite plus quite a bit of new functionality.

The Clarion Magazine Countdown Contest
(Feb 1,1999)

The Unofficial Clarion OOP Page
(Feb 6,1999)

From The Publisher
(Feb 8,1999)

Feature: ABC or Legacy
(Feb 8,1999)

Feature: The Clarion Open Source Project
(Feb 8,1999)

The Clarion Advisor: Speed up your APP debugging with a PRJ
(Feb 8,1999)

Interview: Roy Rafalco (Part 1)
(Feb 15,1999)

Feature: Don't Know, Do Care - A Philosopher Looks At OOP
(Feb 15,1999)

Product Review: Xplore Templates
(Feb 15,1999)

David Bayliss On The ErrorClass
(Feb 22,1999)

February News
(Feb 22,1999)

The Novice's Corner - Getting A Grip On Clarion
(Feb 22,1999)

The Clarion Advisor: Topspeed Driver Error Codes
(Feb 22,1999)

Installing Xplore

Xplore is delivered as a zip file containing several Clarion source and template files, text translation files, icons, and a text file. Go directly to the text file (xplore.txt), as this will tell you where to install the various files. All the source code is included.

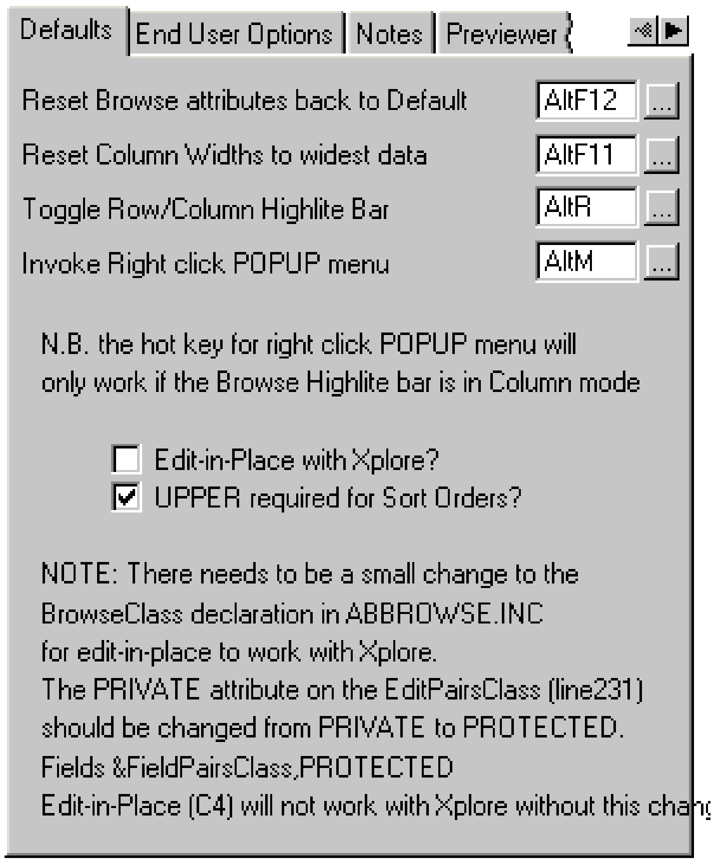
The documentation is a bit sparse, but it's also concise and generally sufficient to the task. You'll need to manually copy the files to the directories specified in xplore.txt – class and translation files to LIBSRC, templates to TEMPLATE, and icons to IMAGES. A proper install program and Windows help file would be a nice touch, but it's hard to complain about this in a product that otherwise offers so many features.

Using Xplore

Using Xplore with an ABC browse is simplicity itself. Simply register the Xplore template chain, open the application, and begin by registering the global template extension (XploreOOPGlobal). This template is used to set the global defaults for any browse boxes to which Xplore is applied. Most of these defaults can be overridden on a browse-by-browse basis, although several are only available here. These include hot keys for applying automatic column widths, resetting the browse back to its default appearance, toggling the highlight bar to cover the whole row (what most of us are used to) or just the current column (spreadsheet style), and invoking the popup menu, which is the primary means of using Xplore's power.

The global extension defaults tab is shown in Figure 1.

Figure 1: The Xplore global extension default tab.



Reset Browse attributes back to Default	AltF12
Reset Column Widths to widest data	AltF11
Toggle Row/Column Highlite Bar	AltR
Invoke Right click POPUP menu	AltM

N.B. the hot key for right click POPUP menu will only work if the Browse Highlite bar is in Column mode

☐ Edit-in-Place with Xplore?
☑ UPPER required for Sort Orders?

NOTE: There needs to be a small change to the BrowseClass declaration in ABBROWSE.INC for edit-in-place to work with Xplore.
The PRIVATE attribute on the EditPairsClass (line231) should be changed from PRIVATE to PROTECTED.
Fields &FieldPairsClass,PROTECTED
Edit-in-Place (C4) will not work with Xplore without this chan

Note that there are some special instructions regarding C4 and edit-in-place. Well-behaved third party products should not require you to modify Topspeed source files, since your changes may be lost the next time you upgrade Clarion. Xplore meets this criterion, with one exception. If you're using C4 and you want to use edit-in-place, you’ll need to find this line in abbrowse.inc:

```
Felds &FieldPairsClass,PRIVATE
```

and change it to

```
Fields &FieldPairsClass,PROTECTED
```

in order to avoid a compile error. This is only necessary if you’re using C4, and isn’t a big concern since when you upgrade to C5 it won’t matter that you lose the change – in C5 the Fields reference is already PROTECTED and Xplore will work just fine.

To use Xplore with a specific browse, bring up the procedure extensions list (either by clicking on the extensions button or by choosing Extensions from the procedure’s popup menu on the AppGen main window). Highlight the browse to which you want to apply Xplore and click on Insert. Choose the XploreOOPBrowse extension. (You will also see the XploreOOPList extension – this is for hand-coded list boxes, not for ABC browse boxes.)

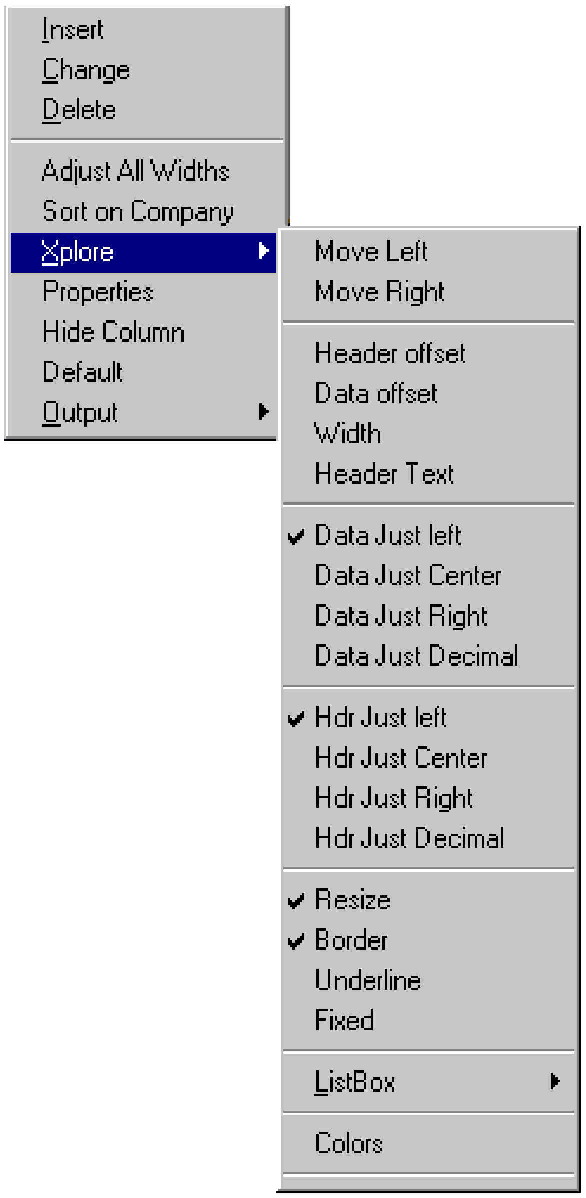
At this point you can go ahead and run your app and try out some of the nifty Xplore features. Bring up the browse and click on a column heading; the browse sorts on that column. Click again on the same column, and the browse sorts on the column in reverse order. For a file loaded list with everything in memory that’s not necessarily a big deal, but this is working with a page loaded list box which means that as you scroll down and more records are retrieved, they retrieved in the order you’ve specified by clicking on the column header. Any filters which have been applied (either specified in the code, or obtained by QBE) remain in effect. All that’s changing is the sort order. Specific fields can be disabled, if you like, and this may be advisable for large data sets where you don’t already have a key on the field. Xplore is telling the ABC BrowseClass which sort orders to use, and the BrowseClass is ultimately passing this responsibility on to the view engine. If you ask the view engine to sort on a field for which it can find no usable key, it will build an index, and on a large file that can take quite a bit of time.

In addition to clicking on columns you can reposition them using drag and drop. Just grab the header and drag it left or right. You can only drag one column at a time, and only to the left or right of the adjoining column, so if you want to move it more than one column you have to drag multiple times. There is another way to deal with this as I'll discuss shortly.

Xploring Popup Menus

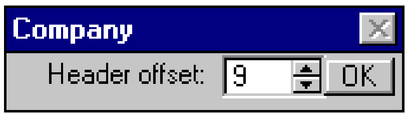
Xplore adds a number of items to the standard browse popup menu. Figure 2 shows a typical popup.

Figure 2: The Xplore popup menu options added to the standard browse popup.



If you choose Header Offset from the menu, you'll see a small dialog like the one shown in Figure 3.

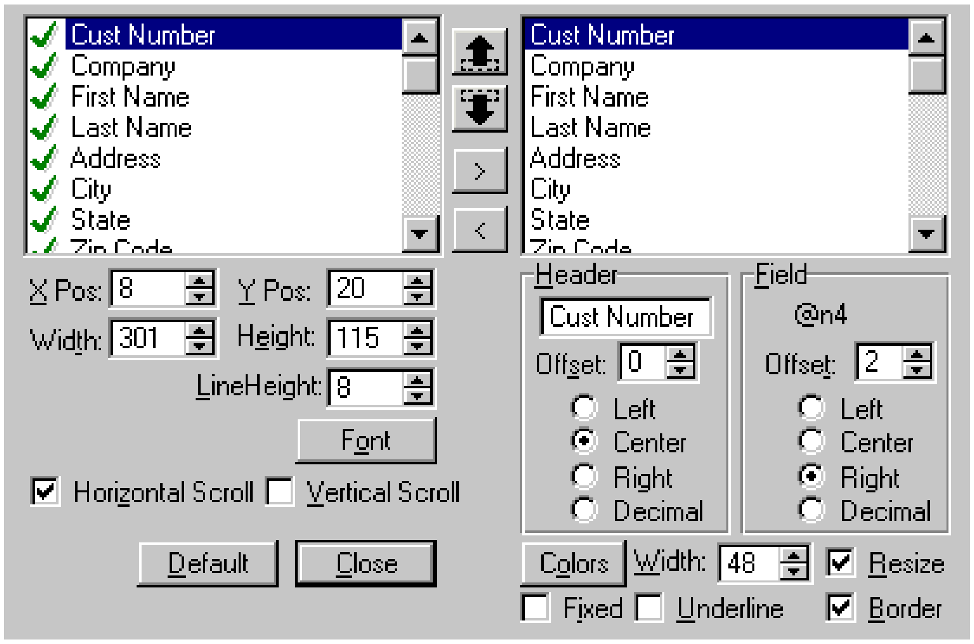
Figure 3: Dialog box for updating browse attributes.



You can adjust the value in the spin box and see the change on screen immediately – the header will shift left or right as specified. Note that this works best with the arrow keys, as the accept loop does not continuously post NewSelection events if you hold down the spin button down with the mouse.

As the menu shows, you (and your users) can apply a wide range of formatting options to a browse, including background/foreground colors, underlining, and text justification. If you want to adjust a whole lot of things at once, choose the Properties popup menu option. This brings up the Xplore Properties window, shown in Figure 4.

Figure 4: The Xplore properties window lets you set all properties at one time.



✓ Cust Number
✓ Company
✓ First Name
✓ Last Name
✓ Address
✓ City
✓ State
✓ Zip Code

Cust Number
Company
First Name
Last Name
Address
City
State
Zip Code

X Pos: 8 Y Pos: 20
Width: 301 Height: 115
LineHeight: 8

Header: Cust Number
Offset: 0
( ) Left [✓ Center] ( ) Right ( ) Decimal

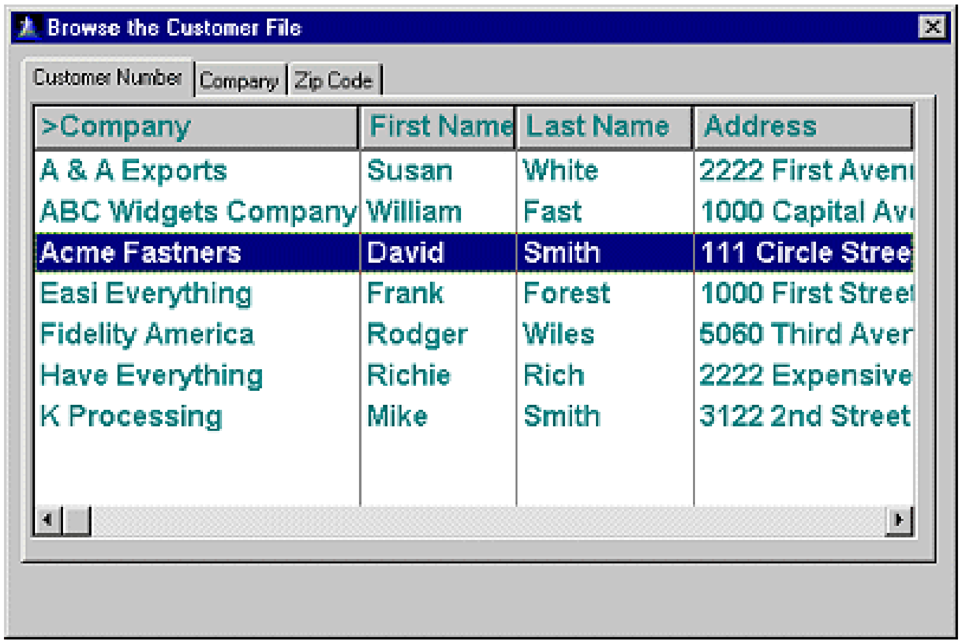
Field: @n4
Offset: 2
( ) Left ( ) Center [✓ Right] ( ) Decimal

[✓ Horizontal Scroll] ☐ Vertical Scroll

Width: 48 [✓ Resize]
☐ Fixed ☐ Underline [✓ Border]

From here you can change just about anything you like on the list box, including the font. If you adjust to, say, Arial Bold 12, in Teal, you get something like Figure 5.

Figure 5: A list box with a large, colored font.



Browse the Customer File

>Company	First Name	Last Name	Address
A & A Exports	Susan	White	2222 First Aven
ABC Widgets Company	William	Fast	1000 Capital Av
Acme Fastners	David	Smith	111 Circle Stree
Easi Everything	Frank	Forest	1000 First Stree
Fidelity America	Rodger	Wiles	5060 Third Aver
Have Everything	Richie	Rich	2222 Expensive
K Processing	Mike	Smith	3122 2nd Street

Just tell them it's Visual Basic, only it works.

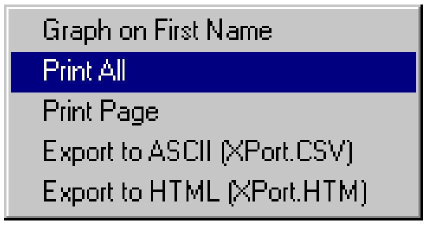
If you do this exercise, you'll need to adjust the line height (which you can do from the properties window). When you do this you change the number of lines that can be displayed on the window, which alters the number of records which the browse will keep in its queue. The browse then needs to be refreshed to display the correct number of rows. All of this is done by the ABC BrowseClass, under the command of Xplore.

There are a number of other nice formatting touches, including the ability to hide columns, to set the width of the column to the widest item of data (of those items presently displayed on screen), and to easily restore all defaults. You can also choose whether to allow different formats for each tab, where you have multiple sort orders, or to use the same format for all tabs. You can choose from several column header indicators (the marks that indicate which column is the active sort order, and whether it is ascending or descending) and you can optionally display a message if the user clicks on a header that isn't sortable.

If your browse has multiple sort orders on tabs (which is what the AppGen will create for you) you can also have Xplore drive the tab selection. If the user clicks on a particular column, you can activate that tab (which may have tab-specific controls such as a button for child records).

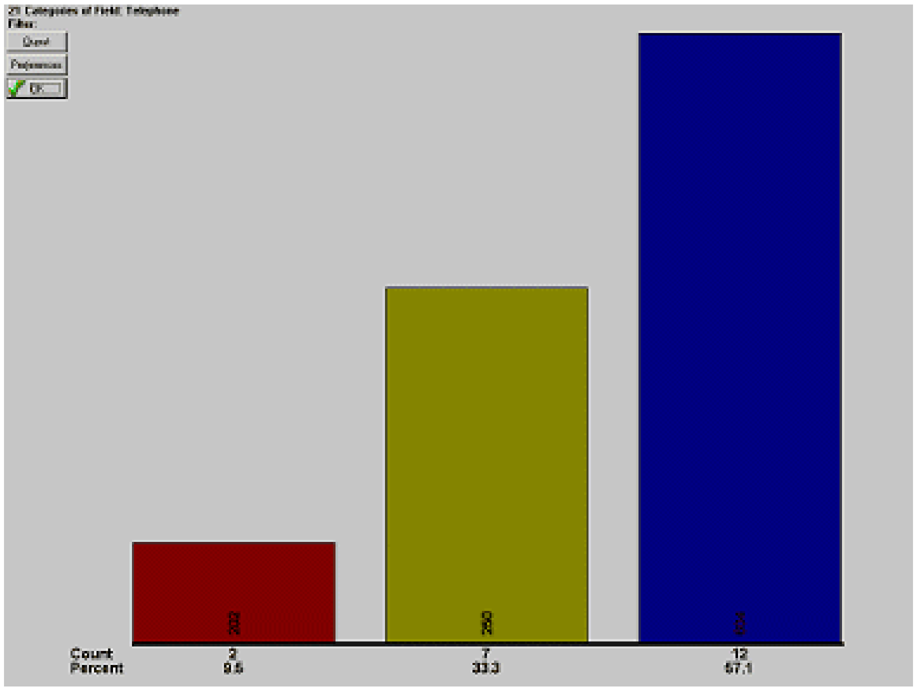
Xplore can do considerably more than reformat browses. On the popup menu is a submenu called Output (see Figure 6).

Figure 6: The Xplore output submenu.



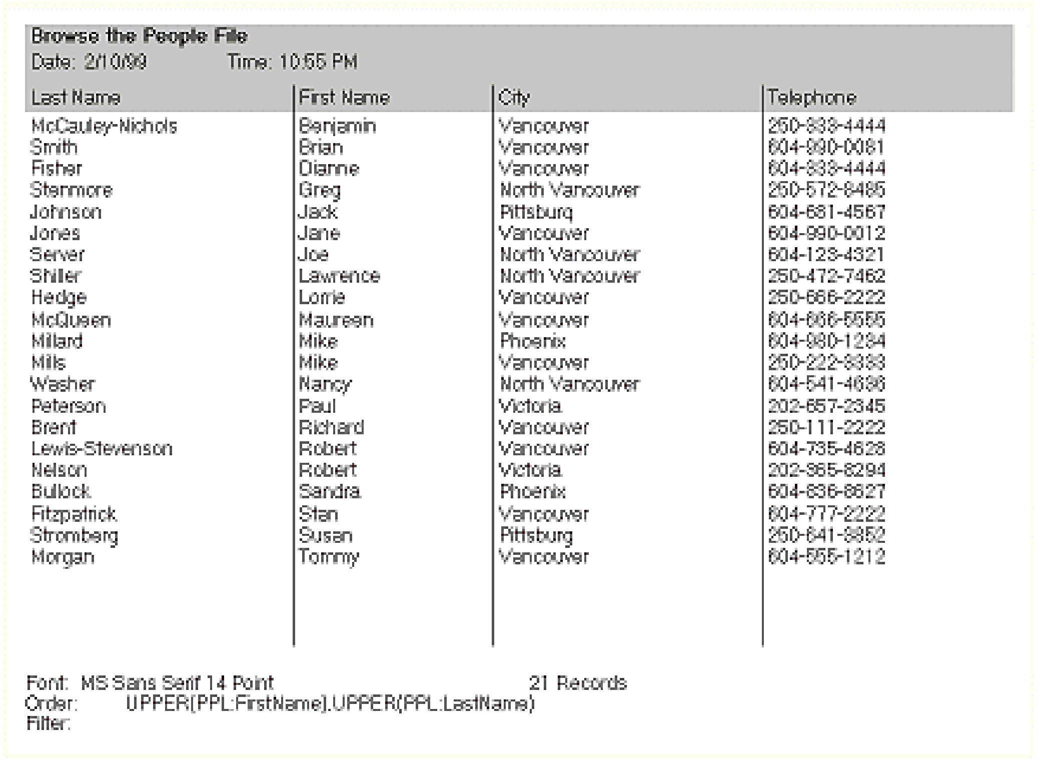
Output lets you create graphs, reports, and other data based on the formatting options the user has chosen using Xplore. Graphs are simple bar charts which by default will have one bar for each unique value in the column. You can, however, specify ranges for the graphs by clicking on the extension's Columns tab and inserting a column field assignment. The column field assignment button contains a button for graphing details where you can assign numeric, date, time, and string ranges.

Figure 7: An Xplore graph of telephone number distribution, by area code.



You can also use Xplore to create mini-reports, again based on whatever fields and formatting options the user has chosen. These reports can be run on the entire data set or just those records shown on screen.

Figure 8: An Xplore report, using the on-screen format.


Browse the People File
Date: 2/10/99 Time: 10:55 PM

Last Name	First Name	City	Telephone
McCauley-Nichols	Benjamin	Vancouver	250-333-4444
Smith	Brian	Vancouver	604-990-0081
Fisher	Dianne	Vancouver	604-333-4444
Stenmore	Greg	North Vancouver	250-572-8485
Johnson	Jack	Pittsburg	604-681-4567
Jones	Jane	Vancouver	604-990-0012
Server	Joe	North Vancouver	604-123-4321
Shiller	Lawrence	North Vancouver	250-472-7462
Hedge	Lorrie	Vancouver	250-666-2222
McQueen	Maureen	Vancouver	604-666-5555
Millard	Mike	Phoenix	604-980-1234
Mills	Mike	Vancouver	250-222-3333
Washer	Nancy	North Vancouver	604-541-4636
Peterson	Paul	Victoria	202-657-2345
Brent	Richard	Vancouver	250-111-2222
Lewis-Stevenson	Robert	Vancouver	604-735-4628
Nelson	Robert	Victoria	202-365-8294
Bullock	Sandra	Phoenix	604-836-8627
Fitzpatrick	Stan	Vancouver	604-777-2222
Stromberg	Susan	Pittsburg	250-641-3852
Morgan	Tommy	Vancouver	604-555-1212

Font: MS Sans Serif 14 Point 21 Records
Order: UPPER(PPL:FirstName),UPPER(PPL:LastName)
Filter:

Two additional output options let you export data to other applications. Export to ASCII will create a CSV file and will (typically) load it into Excel (or whatever other program you have registered to receive CSV files. Export to HTML will create an HTML table of your data, like the one in Figure 9.

Figure 9: An HTML table generated by Xplore

Company	First Name	Last Name	City	State
ABC Widgets Company	William	Fast	Chicago	GA
Acme Fastners	David	Smith	Durham	GA
A & A Exports	Susan	White	Grand Prairie	LA
K Processing	Mike	Smith	Forest Hills	SC
Fidelity America	Rodger	Wiles	Glen Burnie	LA
Easi Everything	Frank	Forest	Louisville	MS
Have Everything	Richie	Rich	Evanston	GA

The Demo

A C4/C5 Xplore demo is available for download, and it has all of the capabilities described in this review. You can get it at the Xplore order page:

http://www.bmtmicro.com/catalog/xplore.html.

Look for the link under the Product Info header.

Unzip the demo to any directory you like as it's a locally compiled EXE with the necessary TPS files. You might want to print out the notes window that appears when the demo starts, as it contains useful information on Xplore's capabilities and can be a helpful supplement to the documentation, particularly with regard to graphing capabilities and using Xplore in parent-child browse pairs.

There are a few minor cosmetic glitches in the demo. The notes seem to indicate that the BrowseCustomer example already has different formats for each tab, while you have to modify the formats yourselves. An appropriate INI file included with the demo would solve this problem, although the program would have to be modified to use a local INI file rather than the Windows directory, which is where it now creates its xplrdemo.ini. As well, on most browses the display string on the lower left corner is not anchored to the bottom of the window and overlaps other window controls when the window is resized.

Support

The author's address in the notes is shown as 75310,1147@compuserve.com, which is incorrectly formatted. Change the comma to a period (the comma is for Compuserve's address format) and it's fine: 75310.1147@compuserve.com

Support is generally via the third part newsgroup at the TS news server (tsnews.clarion.com) or by email to the CompuServe account.

Xplore on the web

The web site is a bit out of date, as it lists Xplore as being for C4 and makes no mention of C5, although the product is definitely C5-compatible. Datamatrix expects to update the page soon.

Xplore Home Page http://ourworld.compuserve.com/homepages/BrianStaff/

On-line Purchase
http://www.bmtmicro.com/catalog/xplore.html

Company Information

Datamatrix Software
4412 East Via Montoya
Phoenix, AZ
USA 8505

BrianStaff@compuserve.com

Summary

Pros

- Easy to implement
- Good control over end-user access to features
- Output to HTML and CSV
- Vast feature list
- Translation support

Cons

- Documentation is a bit sketchy
- Web site is out of date (though soon to be updated, according to the author)
- No install program or Windows help file.

Overall Rating: Highly Recommended

This product is a must-have for anyone who wants to give users a measure of control over the appearance/behavior of ABC browses, or allow quick printing/graphing or export to other applications.

Reviewed by Dave Harms

Reader Feedback

If you have any comments on Xplore or this review, send them to reviews@clarionmag.com.

Do you have a Clarion-related product you'd like reviewed? See our Product Review Guidelines.

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

The ErrorClass

by David Bayliss

Who can understand his errors?
Cleanse thou me from secret faults. Ps 19:12

In Psalm 19 David meditates on the judgements of God and the way that suitable correction can keep the servant profitable. But then he raises the important question, do we always understand the nature of the error? Does the method of correction give us enough information to avoid repetition of the problem? Are we prepared to have the error corrected or would we rather continue as we are? It raises a second issue, the undetected error. Whilst we might happily continue, confidant in our own minds that we are perfect, it is quite possible that faults and defects lie under the surface that are detracting from our overall performance.

Now David was clearly considering issues more fundamental and personal than a computer program but in many ways the construction of the ErrorClass has to reflect many of the concerns expressed in our opening verse. For the ErrorClass to be invoked it is probably the case that a condition or defect has been detected outside of the *will* of the *creator* of the program (you). It is when the chips are down that real quality shines. A chain is only as strong as it’s weakest link. As such the ErrorClass is one of the most fundamental and basic of the ABC classes, it is the one written first and it is at the bottom of the pile (it doesn’t use other classes).

Aim

Consider the external (developer) requirements placed upon it :

1. It must be able to function in a hostile environment (it will be called *because* other parts of the ABC system are sick). This means it can’t be too flash (showy/gaudy) and should assume the minimum.
2. It should be able to interact with the end-user when required.

The Clarion Magazine Countdown Contest
(Feb 1,1999)

The Unofficial Clarion OOP Page
(Feb 6,1999)

From The Publisher
(Feb 8,1999)

Feature: ABC or Legacy
(Feb 8,1999)

Feature: The Clarion Open Source Project
(Feb 8,1999)

The Clarion Advisor: Speed up your APP debugging with a PRJ
(Feb 8,1999)

Interview: Roy Rafalco (Part 1)
(Feb 15,1999)

Feature: Don't Know, Do Care - A Philosopher Looks At OOP
(Feb 15,1999)

Product Review: Xplore Templates
(Feb 15,1999)

David Bayliss On The ErrorClass
(Feb 22,1999)

February News
(Feb 22,1999)

The Novice's Corner - Getting A Grip On Clarion
(Feb 22,1999)

The Clarion Advisor: Topspeed Driver Error Codes
(Feb 22,1999)

3. It should quickly and cleanly execute any corrective action required
4. The developer should be able to add or tailor any error message with a minimum of fuss (especially ‘foreign’ languages <such as American!>)
5. The developer should be able to tailor the corrective action taken
6. The developer should be able to tailor the error screens presented.
7. It should be easy to throw an error message.
8. It should allow the separation of error recording and reporting when required. This one needs explaining. Consider an ABC method that is performing some action inside a ‘Try’ style method (eg FileClass.TryFetch). It encounters some error that it wants to *register* with the error class. It then returns to its’ caller the error flag. The caller then needs to be able to find out what kind of error happened, and in some cases cause the user to be informed. Now the caller cannot just re-raise the error (as the global error state will have changed). So the error class needs to be controllable with regard to which errors it records and which it doesn’t. Because it is possible for errors to be recorded yet not externally detected many of the other ABC methods return an error state. This is particularly noticeable within the FileManager class.

Now on top of the actual requirements placed upon the class when written, there is a *hidden agenda*. Put another way, I like to keep my options as open as possible for as long as possible. There were three extra issues I wanted taken into account :

. Parts of the CW library may wish to use error managers without ‘corrupting’ the error manager seen by the user. Therefore although the templates will probably have a global error manager, the base classes shouldn’t assume it.

b. The global error reporting variables (ERROR() etc) are really getting long in the tooth in an environment where multiple files are being accessed. Future file drivers may want to report back error conditions using more modern methods (such as the BUILD command sending ‘building’ events). It would be nice to write the fileclass in such a way that the library can move forward without breaking the developers programs.

c. I believe exceptions should be the exception. I don’t want every third line of code reading IF Something_Horrible_Happened THEN DoThis .

Design Considerations

The aims are always there to depress. When reading that kind of spec, I look out for repeated words, especially ominous ones. Looking at the above the word is ‘tailor’. It means everything has to be soft. No quick hacks or hard-coding. We are writing an engine, not a solution.

The next thing to consider is ‘what is on my side’.

Number one is that although the ErrorClass must be *light* (not use too many resources when dormant) it does not have to be that efficient (it being invoked will typically result in user interaction). Further there will probably not be that many of them floating around so memory consumption is not a *horribly* big issue.

The next thing to consider is who is going to be use the various parts of the specification and how. It is the insight at this point that leads to the design of the class. If you look at the above the class falls into two very distinct usage camps, you and me. More accurately, 1, 7, 8, a, b & c are mostly the concern of people writing ABC classes (or extensions). 3,4,5,6 (possibly 7, although templates help here) are the concern of the application employers. 2 is the concern of the end user, any problems here I can blame someone else for because of 3,4,5 & 6. This duality of problem means we may split the method of solution. This may or may not help, but at this stage we're looking for anything that may help.

Also you need to consider what *level* of developer will wish to make use of this functionality. Is it some baroque tit-bit to satisfy the bit heads, or day by day functionality to be used by everyone. In this case 3,4,5 definitely fall into the latter camp. This means we need a no-coding solution. The part of the solution for 'ABC' coders can be tougher if needs be.

Another plus is that *most* people will be content to do 3-6 at compile time so we can make that a restriction of the *easy* way, run-time tailoring can call for more work.

The Angle

In any given class, I like to have a unifying theme or concept that makes the class **a** class (as opposed to a collection of classes). In this case the whole class hangs around the .trn file. I will explain the structure of this file presently, however the notion is simply this. Define all errors and actions in a data item (or structure). This structure can clearly be altered and therefore all the required tailoring can be done at run-time (code tends to need compile time modifications). The defaults for this structure can be stored in a simple ascii file (the .trn) which can be safely tackled by any developer. Further the .trn is independent of both the class implementation and interface and can therefore remain constant throughout future releases. You may almost think of the .trn as a program (in a peculiar language) that is *interpreted* by the error class. Hey! Interpretation is slow! Yup, my design considerations say that is ok.

So what data actually needs storing for each error? We choose four :

1. An id to identify the error (these are defined in aberror.inc so they are available wherever the error class is used).
2. A severity level. The aim of this is to give a 'first stab' at the required corrective action. The values cover 'do nothing' (Level:Benign), through 'close the app' (Level:Fatal). In between there are options to question the user for a response (Level:User)
3. The next field is the title to be used on the user interaction dialog
4. The final field is the text to be used on the user screen. This field has some *magic*, namely it can contain macro (or substitution) values. These are detailed in our manuals.

Now I eventually came up with a clever method of storing & using this data, it is detailed under the AddErrors(ErrorBlock) method below.

The Implementation

Probably the way to describe the implementation of the class is simply to talk through the various methods. I am using the C5 sources. There are two obvious orders to use, either alpha or the order in the source file. I'm going to use a third, the order they build upon each other. I assume you have these sources to hand and will not repeat information contained therein.

ErrorClass.AddErrors PROCEDURE(ErrorBlock ErrsIn)

This method is really the heart of our implementation and trn file methodology. In order to work upon our data list we eventually want it stored in some easy data format (we choose a queue). We don't really want the user of the class to have to pre-fill the Q (or even know it exists) so the logical thing to do is have an AddError method that takes the four data types as parameters. The problem is this makes the code to initialize the error manager long and tedious. It also means that to alter how the ErrorManager works would require altering a code file. Further our executable would end up with two copies of the data (one to initialise, one stored). It would also mean imposing an upper limit on the length of an error message.

About Three O'Clock in the morning I had a brain-wave that later became central to the way ABC handles constants (embodied in the ConstantClass). It essentially uses a low-level detail of Pstrings, that the length of the string can be intuited from the data. Using this fact, and known details of the layout of bytes and shorts, it is possible to step through a constant group at a 'byte by byte' level and select out the data items.

This method therefore does precisely that, it contains an extra trick that prevents the strings being duplicated (it is this trick that stops us re-using the constant class). Rather than copy the strings into the queue, we have string references in the queue and simply assign them to the string slice that already exists within the group. This drastically cuts down memory consumption and removes any length limitation imposed by the queue (the pstring is limited to 255 characters of course).

NB: There is one hidden gotcha though. It means that if you do an AddErrors call from within a procedure using a group that is local data you *must* have the static attribute on the group if that error id is going to be used outside of that procedure (ie once that procedure scope has gone).

ErrorClass.SetId PROCEDURE(USHORT Id,UNSIGNED StartPos)

This method implements the next key feature, finding the correct data record for a given message Id. This is not just a queue get because we want to allow scoping. What I mean by that is we want to be able to alter the behaviour of a given error message within a limited piece of the program execution. We do this simply by saying that AddErrors always appends and that SetId always searches backwards to find the matching Id. The two parameter form is to allow the search to start in mid-list (used by the %previous macro). The one parameter (more common) form does a search from the end of the list.

The Takes

The Take methods are used at the point some form of 'user interaction' is required. `TakeError` is really the despatcher, it simply calls one of the other Take methods that correspond to the differing error levels available. Now when embarking upon a group of methods, such as the other Take methods my mind automatically does a form of 'parent-hunting' or 'common denominator finding'. If you have to write 5 similar methods, find out why they are similar and common up the code.

Each take method is responsible for taking the data for its' error message, substituting the macros with values from the class' data and then displaying the results and acting upon the user response. The 'action' is dependent on error level, the substitution isn't, so a SubString method was created to handle the common work. The Take methods then become fairly trivial. `Message` was used in preference to a window to comply with requirement 1.

There is one ‘extra’ take method, `TakeOther`, this is really there for internal / 3rd party usage. Essentially it allows for new error levels to be defined which will then be re-routed to this method. It corresponds to the `ELSE` clause of the `TakeError` method.

Requirement 6 is met by making each `Take` method virtual. `TakeError` does not need to be virtual as it is not called from ‘underneath’ only from on top.

ErrorClass.SubsString PROCEDURE

This procedure performs the mapping from a string with macros to a string with the macros substituted. This allows all the private data inside the class to be code at, but via an interface. This satisfies requirement b. If errorcode et al begin to disappear Substring will become more complex, but user code is preserved. The beginnings of this are happening already as ErrorText is replaced with FileError rather than Error for an errorcode 90. For reasons of efficiency, the SubString method assumes the Id has already been selected using SetId

Note too the %Previous macro, this allows the user to append a message to the one generated by the system. Up to C5EE these macros have been case sensitive, we will probably change this for a future release.

ErrorClass.SetErrors PROCEDURE

SetErrors is the main routine to implement requirement 8. If the `Takes` report the errors, then the `SetErrors` records them. Note too SetFile & SetMessage, these are simply providing additional information that the error class might find useful. At first sight the might seem an overly wide interface, why not set all the ‘error state’ in one shot? This is really a concession to usage. This form of separation allows, for example, the file class to ‘pre-stuff’ information into the error handler *that may or may not be used later*. Thus the FileClass SetThread routine (called at the start of every file class action) does a SetFile. This avoids duplication of information and helps requirement c.

ErrorClass.Throw PROCEDURE(SHORT Id)

At this point we have implemented most of the initial requirements, some extra methods were added to make the interface simpler to use (especially with regard to requirement c). The Throw procedure really undoes requirement 8 to fulfil requirement 7. If you like we are now playing the numbers game. First you implement the methods that give you the functionality, then you try to find the way that interface is used 80%+ of the time, and simplify it. In this case it turns out that quite often you want the error processed immediately, throw does this. We also found that SetMessage happened just before the throw quite frequently, hence ThrowMessage.

ErrorClass.SetFatality PROCEDURE(SHORT Id,BYTE level)

SetFatality is a short cut (you can do the same with scoped AddErrors) to allow the action taken on certain errors to be ignored (or escalated). Thus if you wished to turn off the ‘NoRecords’ message around a report you can simply do

GlobalErrors.SetFatality(Msg:NoRecords,Level:Benign)

In the opening embed of the report. (You need to reset the fatality level unless you want the change to be permanent).

ErrorClass.RemoveErrors PROCEDURE(ErrorBlock ErrsIn)

I left RemoveErrors to last because it in one of those rare instances of a method that is completely mis-coded and mis-designed purely to make using it easier. Logically RemoveErrors should work on a list of Ids, or an individual Id. The only information you need to remove an error is the index of it. However, the time you use RemoveErrors it will almost certainly be paired with an AddErrors call, you are effectively defining an error scope. In fact the AddErrors will nearly always be at the head of the procedure, the RemoveErrors at the end. Now, if you have an AddErrors you want to make the RemoveErrors match ‘perfectly’. So having two different data streams is dangerous. Further the ErrorBlock used by AddErrors is still available (remember it has to be static) so the easiest thing to pass in is the same ErrorBlock again. Thus if you look at the implementation you will find most of the procedure is simply skipping over the information it doesn’t want, finding the Id, and then deleting that field.

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.

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

clarion magazine
Good help isn't that hard to find. $6.25/month

Feature Article

February, 1999

February 22, 1999

Call For Open Linux Project

Ron Schofield has placed a call in the topic.chat newsgroup on the TS news server (news://tsnews.clarion.com) proposing a collaborative project which includes the following goals: "a) We create a template that takes an existing Clarion .app and generates C code that can be compiled into a Linux program. b) We create file drivers to access the data that we create in Windows apps." Ron's message has generated considerable discussion, and the web site is now up.

CopyFlash 1.2 Released

Sterling Data has released CopyFlash 1.2, a template that places a Copy button on a browse, with the following extra features:

- Copy all related child records. The developer can choose which ones to include.
- Two operation modes: Open the update form before copy or just do a straight copy without editing.
- Copy within the same file or copy to another set of parent/child files. Autoincrement is automatically handled by both modes.
- Actions to take after a record is copied: 1) Delete the copied record or 2) Set a flag field in the copied record such as an archived flag or 3) do nothing.
- What text (if any) should appear on the popup menu can also be customized.
- Option to display a message such as "Record Copied OK" - useful when copying to another file and its not immediately obvious a copy has occurred. It reassures the user something has happened

CopyFlash costs $29 and there are no runtime royalties. All source code supplied. A demo is on the web site: http://www.sterlingdata.com

Wise Support Now On TS News Server

The Wise support newsgroup for Wise for Clarion is now being carried on the TS news server (tsnews.clarion.com). Update your newsgroup list.

TimeSaves Gizmos Updated

An update to TimeSavers Gizmos!, Version 2.2 has been released on 2-20-99. It is available for immediate download at www.clarioncentral.com. This latest version now features support for Sliding Controls. A Sliding Control is similar to the sliding Volume Control in Windows. This gives developers the ability to easily incorporate these types of controls into their own applications. There is a demo available now, at www.clarioncentral.com.

TimeSaver Gizmos! is a unique collection of Templates that allows you to quickly enhance your applications for your end-users. Selected features include:

- Have a Field automatically validate, even if the user left it blank (just like the good old DOS days). Will NOT validate when the Cancel button is pressed!
- Have your List Boxes be "sizeable" by your End-Users (sliding list boxes)
- Allow your End-Users to change the placement of controls on the screen
- Have many Entry Fields share the same Lookup Button
- Quickly export your Dictionary Definitions for End-User documentation
- Convert Numbers to Words (check fill-in)
- Alert Key template lets you alert buttons, tabs, etc. and perform an action you specify
- All Templates work with CW and Powerbrowse (for those still using it!)

For current owners of TimeSavers Gizmos, this update is FREE! Just use the original password to install. If the password has been misplaced, please e-mail john@clarioncentral.com.

Wise Offers InstallBuilder/InstallMaster Training

Wise offers training classes as a one-day QuickStart program, or as part of a two-day Intermediate program. Classes are being offered at select Wise Solutions training centers, and also can be conducted at client locations. Training encompasses two of the developer suites, InstallBuilder and InstallMaster. The one day QuickStart training program ($395.00 per attendee) is a general introduction to the products. The two-day Intermediate program ($795.00 per attendee) includes QuickStart, script editing, file patching, custom dialogs, runtime deployment, and more.

February 17, 1999

Topspeed Hires Watts To Head Sales Effort
Topspeed has hired Frank Watts, formerly VP of Sales for LogicWorks (publishers of ER*win*, an industry-standard database design tool), as Senior Vice President of Worldwide Sales.

Is Pentium III An Invasion Of Internet Privacy?

This News.com article looks at the growing concern over the Pentium III's ability to transmit a unique serial number to Internet sites. Intended to fight fraud, this capability may also allow companies to track where a user has been on the Web.
For more on the Processor Serial Number furor, visit the Big Brother Inside page. (Okay, so technically this isn't a Clarion-related article, but you're reading it on a web site, so...)

February 9, 1999

Wise Solutions Offers Tech Support For Installer

As of Clarion 5, Topspeed has replaced its own Distribution Kit with Wise for Clarion. Technical support for Wise for Clarion is available from Wise Solutions, through their news server. To reach the newsgroups via your browser, go to news://news.wisesolutions.com/wise.wiseforclarion. To set up a news reader, set the news server to **news.wisesolutions.com**. The newsgroup name is **wise.wiseforclarion**.

Read Next Month's News

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

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

Feature Article

February, 1999

The Novice's Corner: Getting A Grip On Clarion

by Dave Harms

For those of us who have been using Clarion for years, the application development environment, with all of its idiosyncrasies, is second nature. If you've just picked up Clarion, however, your initial reaction may be more one of confusion than familiarity.

I don't think that Clarion is any more difficult to grasp than other development environments, but because it has some unique features it doesn't offer as many points of familiarity to the average programmer as, say, VB, Delphi, or most C++ environments. In this article I'll give you a road map to the Clarion development environment, including the use of templates and the "standard Clarion" approach to application design.

The Minimal Clarion Program

Most Clarion developers spend the majority of their time in the Dictionary Editor and the Application Generator, two tools that greatly facilitate the creation of large and complex applications. What these tools ultimately accomplish is simply the creation of files of Clarion source code (which typically have the extensions .CLW or .INC). These source files are then compiled and linked into a working program (EXE) or library (DLL or LIB).

With respect to compiling source code Clarion is not much different from Delphi, or a C++ or Java development environment. Where Clarion differs is that it gives you the ability to create quite large applications without ever *having* to look at a single line of Clarion source code. In practise, most developers do work with some source code, but the majority of the code writing, if you will, is done by the Application Generator, usually in conjunction with the Dictionary Editor. I'll have more to say about those tools a little later.

You might think that all of this code generating power came about because Clarion is a difficult language to work with. Quite the opposite is true. Clarion is a compact and highly readable language. For instance, the classic "Hello World" program in Clarion is brief and to the point. A number of variations come to mind, but one of the shortest is shown in Listing 1.

Listing 1. A Clarion "Hello World" program.

```
program
   map
   end

   code
   message('Hello, world.')
```

The one thing all Hello World programs have in common is that they're completely useless and often don't represent the way real world programs are written anyway. The same is true of this Clarion example. Hello World is contained one source file, or module. A smallish real-world application (say an invoicing program) would probably have from 30 to 50 separate procedures, and you could have one source file for each procedure. And each source module would be considerably bigger than hello.clw.

All of that means that to write an application line by line in a source code editor in the style of hello.clw can be a somewhat painful proposition, even with a highly expressive language like Clarion. Application developers typically look for ways to speed up this process.

There Has To Be A Better Way

One way to speed up your application development is to use a prewritten library, whether it's a procedural library or a class library (for the object-oriented among us). Libraries can save you a whole lot of time. They're not always as flexible as you might like, since you often can't change them, but well-written class libraries in particular can give you many ways of enhancing functionality without altering the original library.

Clarion has always used procedural libraries, and since version 4 has also had an extensive class library. In fact, since your application runs on Windows, one way or another it has to make extensive use of the libraries of code built into Windows. (Usually Clarion developers are shielded from Windows code by Clarion's libraries which act as an intermediary, but if you wish you can interact directly with Windows by using the Windows API.)

Using libraries improves development speed, but doesn't exactly get it racing. To really kick things up a notch you need to look at other technologies.

Speeding Development With Templates

Clarion has had some form of code generation capability since the second DOS version, called Clarion Professional Developer, or CPD. CPD introduced the Designer and the model file. The Designer was a visual design tool that let you create procedures, each of which could have an accompanying window or report on which you could place fields from a data dictionary. You could also add text and draw lines. Don't laugh. Back in the '80s this was exciting stuff!

The model file contained blocks of Clarion code for various kinds of functionality (menus, browses, etc) as well as symbols that could be replaced by information the user specified in the Designer. When it came time to compile, the Designer would read the model file and substitute field names and other information for the symbols, and presto! Instant source code.

There were some problems with this approach. If you wanted to change how Clarion generated code, you had to modify the model file, and you could only use one model file at a time, so it was difficult to mix and match features from different third party vendors. As well most of the decision-making code was internal to Designer, so even if you modified the model file you were under a lot of restrictions as to what you could do. You were seriously limited in the number of places you could add your own code to what Designer created, and any changes you made directly in the generated source would be lost on the next compile.

In the years since CPD Topspeed has improved this code generation technology radically. Designer has been replaced by the Application Generator, the Dictionary Editor is a separate entity, and the model files have been replaced by template files. The result is a far greater level of flexibility.

Fundamentally, however, the Clarion approach to development isn't that different from what it was in the early days. Do as much work as you can visually. Let the AppGen spit out the mundane, repetitious code. Write the code AppGen won't and embed it, or better yet modify or add to the templates or the class library and thereby extend the application development environment to suit your needs. You can, in effect, train Clarion to work the way you work.

Starting With The Dictionary Editor

Most Clarion application development starts with a data dictionary, which defines one or more data files (more commonly called "tables" by the rest of the programming world, and by Clarion developers working with SQL). If you wish you can create an application using the Quick Start method which lets you define a simple app with one data file, but the result is rarely good for much more than a test application because of the severe limitations on the type of file and fields you can create.

The focus on the dictionary editor reflects Clarion's origins as a business software development language. You can certainly write Clarion applications that don't use data files, but most of the time people buy Clarion because they have data they want to create, query, or otherwise manipulate.

Assuming you want to create an application that uses data files, begin by choosing File|New|Dictionary.

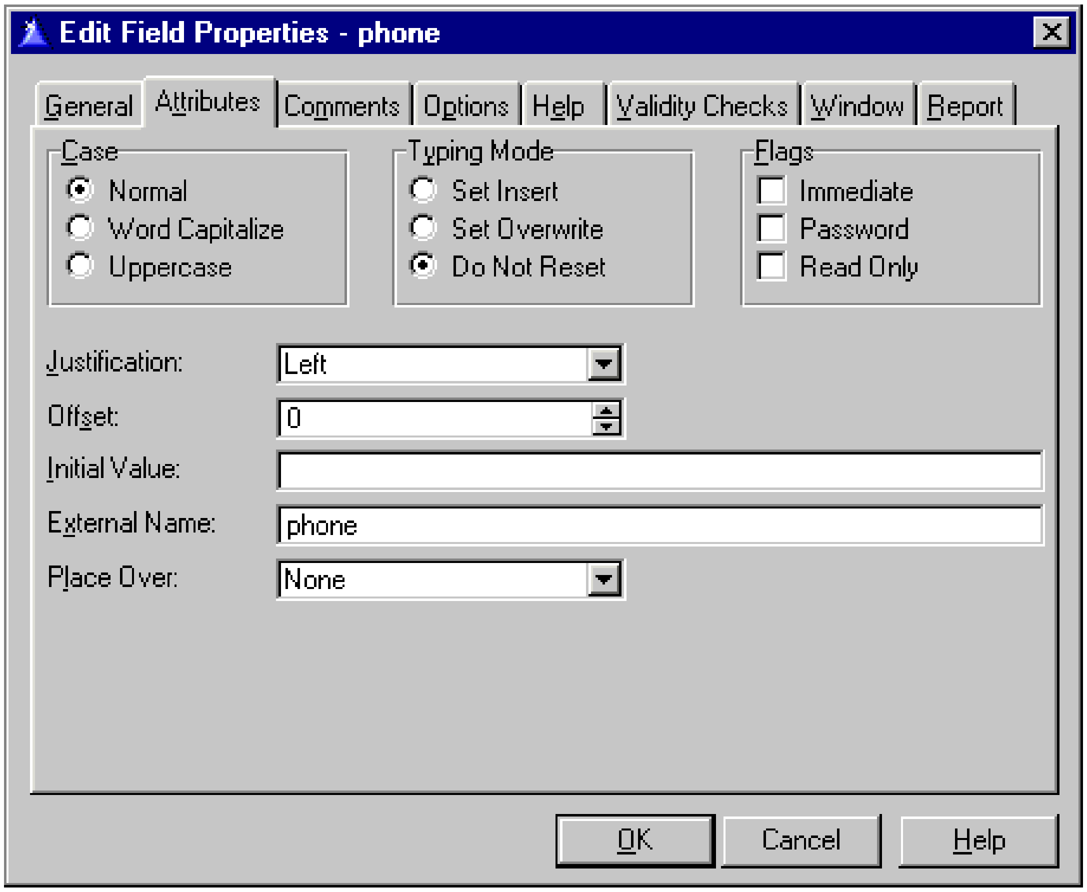
There are three main ways to define data files. You can create the files one at a time using Quick Load. If Quick Load is turned on (see Setup|Dictionary Options, File Options tab) when you click on the Add File button in the Dictionary Editor you have the option of using the built-in Quick Load tool to rapidly specify a data file (you can fine tune this file later). Quick Load only requires you to enter a file name and prefix, a file driver (to suit whichever database back end you're working with), a prefix to make the file's fields unique within the database. For each field you specify a field name, a picture token which indicates how the field should be displayed, and how long it will be, and optionally a key for the field. Quick Load only lets you have a single field per key, but again you can change this later if you wish.

If you decide to not use Quick Load (and after the first few tries, you probably won't), you'll need to specify a bit more information about the file and the fields. Even so a lot of information is set to default values. You'll also almost certainly want to create some keys, which greatly speed access to data and allow you to view the data in different orders.

The third way to define a data file is to import it from an existing database. You can import from any database for which you have a driver. There are a wide variety of flat file database drivers included with Clarion, as well as client/server drivers such as ODBC, Pervasive.SQL, MS SQL, and SQL Anywhere. Clarion Enterprise Edition also comes with a Dictionary Synchronizer that makes it easier to keep your dictionary aligned with the database.

A Clarion dictionary is much more than a repository of file formats. Field definitions in particular can contain a great deal of information that will be used to create the application, including the type of control to be populated on windows, the prompt text, column headings, and whether or not the field should be populated at all. You can specify field validations, and even embed information which controls which custom code you want generated along with the field (though this is a little-used feature at present). Figure 1 shows some of the field validation options.

Figure 1. Data dictionary field validation options.



Case: [● Normal] ○ Word Capitalize ○ Uppercase

Typing Mode: ○ Set Insert ○ Set Overwrite [● Do Not Reset]

Flags: ☐ Immediate ☐ Password ☐ Read Only

Justification: Left

Offset: 0

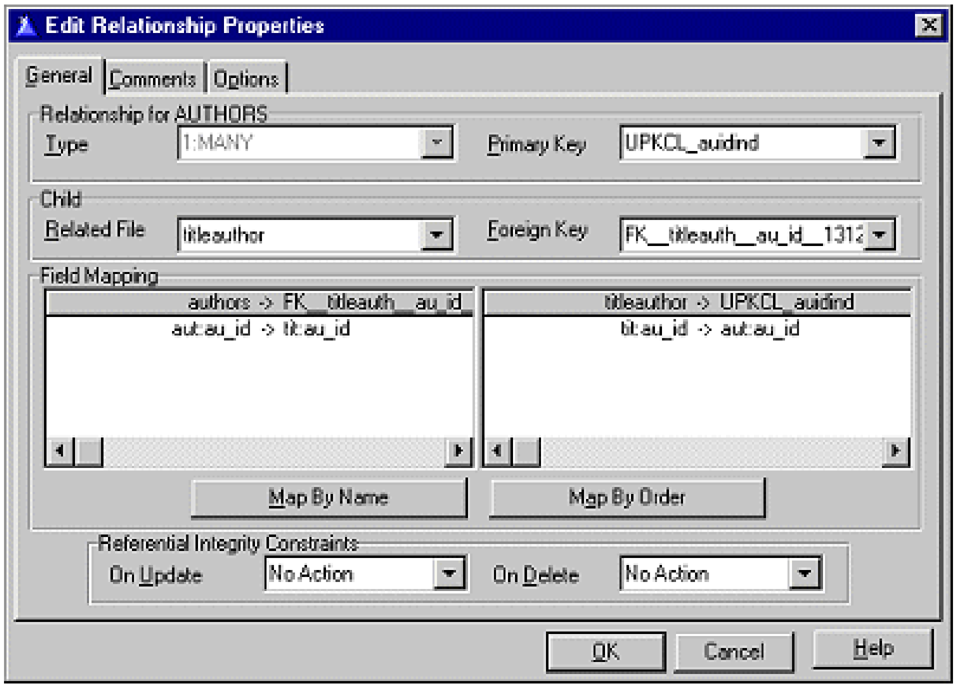
Initial Value:

External Name: phone

Place Over: None

As well, you can define relationships between files in the dictionary editor. Almost all databases are made up of files with connections to each other, such as invoice detail records which belong to a specific invoice header record.

Figure 2. The relationship editor.



Relationship for AUTHORS
Type 1:MANY
Primary Key UPKCL_auidnd

Child
Related File titleauthor
Foreign Key FK__titleauth__au_id__1312

Field Mapping
authors -> FK__titleauth__au_id_
aut:au_id -> tit:au_id
titleauthor -> UPKCL_auidind
tit:au_id -> aut:au_id

Referential Integrity Constraints
On Update No Action
On Delete No Action

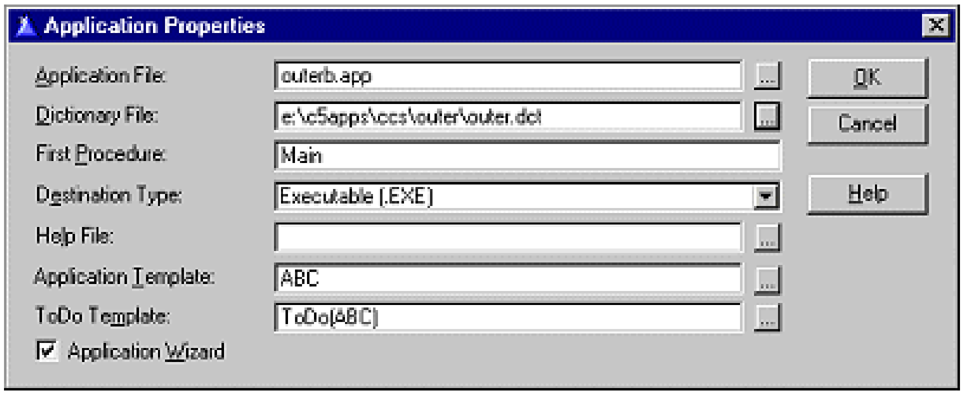
The choices you make in the dictionary editor can have a great impact on the initial appearance of your application, and this will become increasingly important as Topspeed rolls out the Wizatrons, its latest application-generation technology (and the subject of an article by David Bayliss, which will appear in Clarion Magazine in early March, 1999).

In many cases, dictionaries get far less attention than they should because developers are eager to hurry on to the "real" work of creating their applications. Database design is a sometimes neglected aspect of application design, and usually steps missed here will result in considerably more work down the road.

Now Comes The Easy Part

Assuming you’ve created, imported, begged, borrowed, or stolen a dictionary, you’re a couple of mouse clicks and keystrokes away from creating an application. If the dictionary is still open, close it. Choose File|New|Application and enter a name for the application (making sure use Quick Start is *not* checked), and click OK. Next you see the Application Properties window (see Figure 3). Enter the name of the dictionary you created in the Dictionary File field. Accept the default values for the rest of the fields, but make sure that Application Wizard *is* checked. (The Application Wizard is a special type of template that automatically builds procedures out of existing templates, data dictionary information, and some minimal user input.)

Figure 3. The Application Properties window (creating a new application)


Application Properties

Application File:	outerb.app
Dictionary File:	e:\c5apps\ccs\outer\outer.dct
First Procedure:	Main
Destination Type:	Executable (.EXE)
Help File:	
Application Template:	ABC
ToDo Template:	ToDo(ABC)

☑ Application Wizard

Click OK, and the wizard will ask you a fairly small number of questions about which files you want to use, and a bit about the style of the application. After the wizard is done, it creates your application.

But what kind of application is it? How does Clarion know what you want to create?

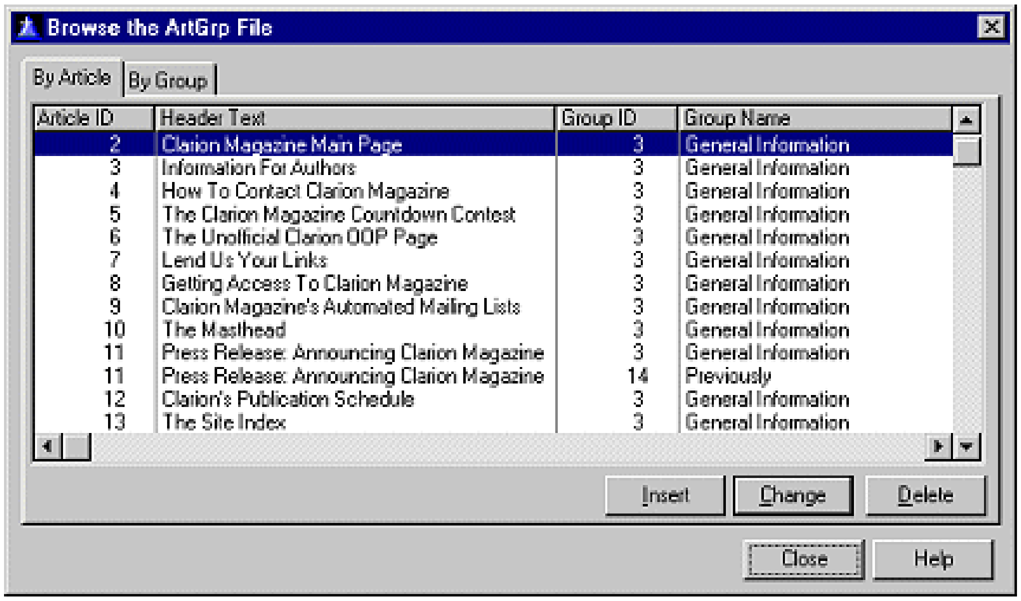
Truth to tell, Clarion doesn't know what you want to create. It does make certain assumptions. One of these is that you want to create an MDI application, with a standard application frame which will contain all of the windows you may want to display, and a menu bar from which you can invoke these windows (each of which is contained in a procedure).

Another key assumption Clarion makes is that you wish to use the "browse-form" paradigm.

The Browse-Form Paradigm

In the browse-form world, you have choices on the main menu that allow you to bring up browse procedures. This means that the first time you see existing data from a database in a typical Clarion application, it will be in a scrolling list, or browse, as shown in Figure 4.

Figure 4. A typical browse.


Browse the ArtGrp File

Article ID	Header Text	Group ID	Group Name
2	Clarion Magazine Main Page	3	General Information
3	Information For Authors	3	General Information
4	How To Contact Clarion Magazine	3	General Information
5	The Clarion Magazine Countdown Contest	3	General Information
6	The Unofficial Clarion OOP Page	3	General Information
7	Lend Us Your Links	3	General Information
8	Getting Access To Clarion Magazine	3	General Information
9	Clarion Magazine's Automated Mailing Lists	3	General Information
10	The Masthead	3	General Information
11	Press Release: Announcing Clarion Magazine	3	General Information
11	Press Release: Announcing Clarion Magazine	14	Previously
12	Clarion's Publication Schedule	3	General Information
13	The Site Index	3	General Information

Clarion browses are a bit different from some you may be familiar with in that they are (traditionally) page loaded. Only the records you see on screen have been loaded into memory, and if you scroll past the last or first record on screen, the next or previous record is retrieved (if possible) and the record that scrolls off screen is discarded. This is a useful feature when you're dealing with large data sets, although it does introduce some difficulties in managing the scroll bar, since it's more difficult to accurately position a scroll bar to indicate the record's relative position in the entire data set. In Clarion 5 you have the option of creating file loaded browses, in which all valid records are loaded into memory.

To update a browse, you typically click on an Insert, Change, or Delete button (or use a popup menu on the browse) to bring up another window (and therefore another window – you can have multiple windows in one procedure but it's awkward and not recommended) which allows you to update the browse record. In C4 and C5 (using the ABC templates) you have the option of not using the form. Instead you enter data directly into rows on the browse, but the templates still generate forms for your browses.

Forms will also be generated with browses if the file that's being updated has child records, as defined in the data dictionary. The controls used on the form (entry fields, spin boxes, drop combos, etc) are also taken from the data dictionary.

The appearance of a wizarded application is largely determined by what information has been placed in the dictionary, as well as by the overall concept of the browse/form. In general Clarion tends to steer you down the browse/form path. That's not a problem if browse/form is what you want, but if you're more accustomed to starting with a form and letting the user look up records from that form, you may need to do a bit of work or cast about for some suitable third party templates.

Maximum Wizards

You may want to consider starting your development of a particular application by going through the App Wizard process a number of times, recreating the application each time. After each app is created, compile and run, and evaluate the result against your final goal. Explore the dictionary options, fine tuning your files, fields, and relationships to get the automatically created application as close as possible to your desired result. This is particularly important if you expect to be creating multiple applications using the same dictionary, as you will want your applications to share the same look and feel.

When your dictionary has reached a suitable level of maturity, you can either modify your last wizarded application, or create a new application without using the Application Wizard option. In the latter case you begin building your application procedure by procedure, usually by starting with a Frame procedure as the main menu. Some procedure types (such as browses, forms, and reports) have accompanying wizards which you can use selectively to recreate what the application wizard did all at once.

The forthcoming Wizatron technology is going to further refine this process. Wizatrons use style sheets to apply user-defined formatting and functionality options to automatically-generated procedures, so it's entirely possible that at least in the early stages of application development you may be altering data dictionary *and* Wizatron style sheets to create your basic applications.

Beyond Wizards

As you become more proficient in Clarion, you will encounter situations where the various wizards (or even Wizatrons) aren't able to create the kinds of procedures you want, even with extensive modifications to the procedure after the wizard's done.

Fortunately, Clarion provides several levels of template granularity for different kinds of tasks.

The Application Wizard is a template that creates procedures, and in the process calls other procedure wizard templates. Individual procedure wizard templates create procedures by combining non-wizard templates. A simple browse procedure, for instance, is a combination of a Window procedure template and a browse control template. This granularity makes it easy to add a second browse to a window, because that simply means adding another browse control template. Anything the wizard does, you can do too. You can combine various kinds of templates within a procedure to create exactly what you need.

Templates greatly automate the process of building applications, but even the most extensive set of templates (and you can easily integrate your own or third party templates) probably isn’t going to do everything you want to do. There comes a time when you want to modify the behaviour of your application and the only way to do it is by writing some Clarion code. You may not need to know a lot about the Clarion language to accomplish your goal, but the more you know about Clarion code, and the templates, and the overall structure of "standard" Clarion procedures and applications, the more you’ll be able to make your application dance to your tune.

At the beginning of this article I mentioned that any Clarion program is, ultimately, Clarion source code that is compiled into an EXE or DLL (or LIB). The Application Generator creates this code based on the templates you use, and so you might think that the best place to put your source code is in this generated code. The only problem with that idea is the next time you make a change to a procedure in the AppGen and regenerate the code, your changes will be lost. What you need is a way to insert your code into the templates.

This code insertion is done by means of source code embed points. Embed points are under the control of the templates, and they are just about everywhere. If a template is used multiple times in an application or procedure, then each instance of the template gets its own set of embed points.

A Place For Everything

All of the embedded source, as well as all of the information specified by you or by the various wizards regarding the appearance and behaviour of your application is stored in the .APP file created by the application generator. The data dictionary contains the database specification, as well as a great deal of optional information about field appearance and validation. And the templates are actually source files, but in order to be used by the AppGen they must be loaded into the template registry (REGISTRY.TRF, in the template directory). Under normal circumstances you will have both the text versions of the templates and the registry, although at a minimum you only really need the registry.

In addition to the template registry, dictionary editor, and application file, any given Clarion application needs some of the source files in the LIBSRC directory in order to compile, as well as some precompiled binaries in the BIN directory to link and run. To be completely safe on your backups, you should save, at a minimum, the APP and DCT files, and any changed files in the TEMPLATE and LIBSRC directories. The template registry can be recreated at any time with no ill effect on applications, unless a template file (or some part thereof) has gone missing in the interim.

Next in the Novice's Corner – Understanding Template Types and Embed Points.

David Harms is an independent software developer and the co-author with Ross Santos of Developing Clarion for Windows Applications, *published by SAMS (1995). He is also the editor and publisher of Clarion Magazine.*

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

The Clarion Advisor

February, 1999

Topspeed Driver Error Codes

Many Clarion developers use the TPS file format for small- to medium-sized databases, and with good success. Through Clarion release 4A, however, some users reported regular data corruption problems.

With driver fixes in 4B these problems seem to have largely gone away, but some forms of corruption are still outside the control of the driver, and network installations can be a particular problem. Anyone storing TPS files on an NT server with Windows '95 workstations should be sure to install Microsoft's redirector patch. You can download it from:

http://support.microsoft.com/support/kb/articles/q148/3/67.asp

Thanks to Steve Mull for providing the link.

The following table lists the possible Topspeed error codes (thank you to Nigel Hicks for providing this!) as well as an informal count of reports of these bugs in the newsgroups.

If you have any additional comments you'd like to see added to this list, email advisor@clarionmag.com.

Error	Description	Dates Reported
231	Trying to append a record when the btree is marked read-only	
232	Cannot get btree header when trying to append a record	

256	Cannot get btree header when trying to get a record	
300	Btree structure corrupt (discovered when deleting a record)	
327	Btree structure corrupt (discovered when putting a record)	
337	Trying to put a record when the btree is marked read-only	
522	Invalid data size found while unpacking record (from disk)	
530	Invalid repeat count found while unpacking record (from disk)	
706	Btree structure corrupt (discovered when inserting a record)	
780	Btree structure corrupt (discovered when removing a record)	
824	Btree record size to big (on allocation)	
1013	Cannot get btree header (while loading root page)	
1043	Cannot get btree header (while packing a record to disk)	
1163	Trying to create new record ID when the btree is marked read-only	
1164	Cannot get btree header (while allocating new record ID)	
1173	Maximum record ID reached on allocation (probably indicating file corruption)	
1194	Trying to insert record when the btree is marked read-only	5/98 7/98

1203	Trying to remove record when the btree is marked read-only	
1258	Btree structure corrupt (obsolete in C5)	
1364	Btree structure corrupt (discovered while splitting a page)	
1477	Btree page size (from header) does not match size stored on disk , could result in a truncated file, run tpsfix	9/98 11/98 (2) 2/98 (2) 5/98 12/98
1602	Btree unpacked page size (from header) does not match size loaded from disk	10/98
1659	Btree page size increased after packing	
1678	Btree page size larger than maximum allowed	
1735	Btree header corrupt (discovered when trying to calculate disk file size)	
1781	Btree structure corrupt (discovered while shifting btree pages up)	
1891	Btree structure corrupt (discovered while moving btree pages)	
1894	Btree structure corrupt - invalid page parent level (discovered while moving btree pages)	
2172	Reading of btree page from disk failed	10/98 (2) 12/98 1/99 (2) 2/98 5/98 6/98 (2)
2183	Too many files logged out	
2272	Encryption block invalid size (discovered when reading page from disk 32-bit)	

2277	ReadFile Win32 API function failed (when reading page from disk)	3/98 10/99
2286	Encryption block invalid size (discovered when writing a page to disk)	
2328	Invalid internal btree locking mode (on lock (32 bit))	
2352	Invalid internal btree locking mode (on unlock (32 bit))	
2341	SetFilePointer Win32 API function failed (when reading page from disk)	
2361	UnlockFile Win32 API function failed (obsolete on C5)	
2447	16 bit close file failed (1)	1/99
2458	16 bit handle duplicate failed (during commit)	
2460	16 bit close file failed (during commit)	
2476	Invalid internal btree locking mode (on lock (16 bit))	
2519	Invalid internal btree locking mode (on unlock (16 bit))	
2528	Unlock dos function failed (obsolete on C5)	
2572	Encryption block invalid size (discovered when reading page from disk 16-bit)	
2582	Encryption block invalid size (discovered when writing a page to disk 16-bit)	

Clarion MAGAZINE

Clarion Development Resources

published by CoveComm Inc.

clarion magazine Good help isn't that hard to find. $6.25/month

Clarion Magazine Information

December, 1999

Subscription Agreement

By subscribing to this magazine you agree to the following conditions:

The information contained on the www.clarionmag.com web site (hereafter referred to as Clarion Magazine) is the property of CoveComm Inc.

As a subscriber, you are entitled to read all parts of Clarion Magazine at http://www.clarionmag.com for which you have been authorized. Clarion Magazine is an **on-line magazine only**. It is not a printed magazine.

You may keep an electronic and/or print copy of such pages as you wish, in unaltered form, for your own use. This includes PDF files provided as part of your subscription. Other than this, you may not, under any circumstances, reproduce or retransmit the information contained in Clarion Magazine.

Subscribers will also be granted access to private newsgroups on the Clarion Magazine news server. You may not provide the ID and password to the news server to anyone else, or distribute or otherwise make public messages from the news server (other than your own messages) without the permission of CoveComm Inc.

If you are found to have violated this agreement, you agree to the termination of your subscription immediately and WITHOUT refund.

This agreement will be governed by the laws of Manitoba, Canada.

Browser And Compile Manager
(Dec 7,1999)

Clarion's Publication Schedule
(Feb 5,1999)

Advertising Policy
(Feb 7,1999)

Open Source Code Available Now
(Mar 8,1999)

The Masthead
(Feb 15,1999)