



# Clarion magazine

## Volume 1, Number 3 - April 1999

### Issue Index

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

#### [Larry Teames on Reports](#)

Larry Teames begins a regular column on reports in April, starting with the basics and divulging some of the secrets he's learned on his way to becoming the acknowledged authority on Clarion report writing.  
Posted on April 5, 1999

#### [How ABC Handles Multiple Sort Orders](#)

A need for speed (sorts) takes Steve Parker down a winding path to ABC's handling of multiple sort orders.  
Posted on April 5, 1999

#### [Ask The SQL Answer Cowboy](#)

Andy "Cowboy" Stapleton is the acknowledged Clarion SQL guru and a regular presenter at Clarion conferences around the world. His company, Cowboy Computing Solutions, produces SQL templates and classes for Clarion. If you have an SQL question for Andy, particularly if it reflects issues that will be of concern to other Clarion programmers, you can post it c/o Clarion Magazine.  
Posted on April 5, 1999

#### [The Clarion Challenge - Results](#)

Results from the first Clarion Challenge in which Clarion coders looked for the fastest, smallest solution to a programming problem. Things didn't turn out quite the way you might expect...  
Posted on April 5, 1999

#### [David Bayliss on FieldClass](#)

FieldClass is one of the smaller ABC classes, but it's responsible for almost 30% of the reduction in generated code between the legacy and ABC templates. David examines the rationale behind FieldClass and explains the implementation. This extensive article includes a lot of useful information on the care and treatment of ANYs.  
Posted on April 12, 1999

#### [Product Review: The Clarion Class Browser](#)

This essential freeware product from Gordon Smith, now a Topspeed employee, makes life easier for Clarion OOP programmers.  
Posted on April 12, 1999

#### [News Feature: Clarion Goes COM!](#)

Ross Santos offers a tantalizing glimpse of Clarion's upcoming Component Object Model layer. As usual, the TS development team is making this inordinately complex MS technology easier to use. Includes links to COM information sites.  
Posted on April 12, 1999

#### [Knowledge Bases On The Web - Feature Interview](#)

Arnor Baldvinsson, Steve Parker, and Troy Sorzano are the minds behind the three first Clarion knowledge bases on the web. In this feature interview they tell how all three KBs came about and what technology they're using to make these invaluable resources even better.  
Posted on April 19, 1999

[DAB Dissects The Clarion Challenge Results](#)

David Bayliss has analyzed the code the compiler created for each of the Clarion Challenge submissions. This article provides a fascinating insight into what the compiler really does with your code. Part 1 of 2.  
Posted on April 19, 1999

[Search Clarion Magazine!](#)

With this week's focus on internet knowledge bases, this seemed like a good time to announce that Clarion Magazine is now fully searchable, which makes the entire Clarion Magazine site another indexed knowledge base! The search engine is publicly accessible, although of course some searches will return links to subscriber-only pages.  
Posted on April 19, 1999

[How To Waste Time In The Newsgroups](#)

Troy Sorzano lets loose on the inefficiency of newsgroups as a way of finding/transferring Clarion programming knowledge.  
Posted on April 19, 1999

[Clarion Magazine Now Available In PDF Format](#)

After numerous requests from readers, Clarion Magazine is pleased to announce that articles are now available in PDF format as well as HTML. PDFs make for easier archiving and better printing.  
Posted on April 19, 1999

[Andrew's Kitchen - Sliding Into Cookery](#)

The cooking Cajun gets his name in lights (again!) with this feature article. Is it food? Is it code? It's a little of both.  
Posted on April 26, 1999

[April News](#)

All the Clarion news that's fit to print, and then some.  
Posted on April 26, 1999

[DAB Dissects The Clarion Challenge Results](#)

David Bayliss has analyzed the code the compiler created for each of the Clarion Challenge submissions. This article provides a fascinating insight into what the compiler really does with your code. Part 2 of 2.  
Posted on April 26, 1999

[The SQL Answer Cowboy Answers!](#)

Andy Stapleton answers questions about PROP:SQL, stored procedures and triggers, and SQL Anywhere 5.5 vs. 6.0.  
Posted on April 26, 1999

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## Feature Article

### Sorting On The Run (Part I)

#### How ABC Does Multiple Sort Orders

by Steve Parker

It's funny how things work out sometimes.

In Clarion 4, Topspeed introduced two features, Additional Sort Fields and the ability to create controls on the fly, at runtime. Neither amounted to much for me.

I just did not understand the utility of runtime creation of controls. Hide/Unhide and Enable/Disable were perfectly adequate to my needs. Particularly because I was working almost exclusively with Internet Connect at the time and a Disabled control in IC generates no HTML (and, therefore, takes no space on the screen), I was unmotivated to discover the charms of this feature.

Similarly, I never figured out what the Additional Sort Fields prompt offered that dynamic indices did not already offer. Maybe, I thought, this was one of those "marketing" features: something that has no real usefulness but can play prominently in an ad or presentation.

(Later, I learned a critical use for this feature. When browsing a SQL table on a non-unique index, attempts to update a row may be met with fatal errors. By naming another column in Additional Sort Fields, making each row in the browse effectively unique, these errors are avoided. So, this is definitely not a fluff feature.)

As I said, it's funny how things work out sometimes. I now need to do "speed sorts." A speed sort is a user-defined and user-selected sort order on a browse or report. The sort criteria are stored in a file field. Users may, obviously, update the file, adding new sort orders, or changing or removing existing sorts. In the DOS program I will be converting, this is accomplished with dynamic indices.

While perfectly functional, there is a substantial downside to dynamic indices. For starters, each and every call to a dynamic index requires a full build. Also, even if fields in existing keys are used, the key will not be used. Oh yes, the build takes place from disk. On a network, this is ... ah, slow.

Since early in the Clarion for Windows product cycle, we have had a powerful View engine. A browse is no longer filled directly from files, but from a view, which is made up of fields from one or more files. All this happens transparently. Topspeed actually started using views late in the DOS product cycle. Before views, browse queues were filled from disk, which had a negative impact on page-to-page scrolling. Now, views are created from disk and the browse is filled from the view.

The important characteristic of views, in the current context, is that views can be sorted and filtered. Views will automatically use any existing keys when possible. And, sorting/filtering can be done in

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

memory, eliminating disk and network access. With large files, views may not always be "fast" but they will certainly be much faster than building an index from disk.

Moving from DOS to Windows should provide an opportunity to re-think and speed sorts sound like an ideal candidate for both additional sort fields, which affect the view, and creating controls on the fly.

The idea I have is to use the ability to change view orders by reading the file containing the sorts. From the sort file, I would create a tab for each sort. When the user selects one of these runtime tabs, simply pick up the sort string/variable, call whatever methods are required to re-sort the view and invoke the order.

Simple.

Unfortunately, the Additional Sort Fields do not support the use of variables. And, to use file fields, as I can with dynamic indices, I must be able to use a variable. But the templates clearly do support at least the use of strings. So, I see three tasks to be accomplished:

1. How are multiple sort orders implemented?
2. How do I create the required (number of) Tabs?
3. How do I "associate" a Tab to a sort order?

## Multiple Sort Orders

While the template prompt reads "Additional Sort Fields," not "Additional Sort Orders," this feature does support full fledged sort orders (not simply sorting an existing key on additional fields). For example, a tab with no key (i.e., record order) and an expression in Additional Sort Fields is a new order.

However, entering a variable in the Additional Sort Fields prompt will not have the desired effect. It will have no effect at all and result in no error or warning from the compiler. The file will display in record order.

The last statement is true at the time that I am writing this. C5A, probably available as you read this, supports variables in this prompt, so part of this discussion is moot. Moot, because with full support of variables it may no longer be entirely necessary to understand how sorting is affected.

In addition, Dennis Evans has posted a very neat class and template wrapper allowing user-defined runtime sorts (available from [www.cwicweb.com](http://www.cwicweb.com)). The user can choose which fields, in which order to sort the browse. The developer, however, retains control over which fields are available for user selection (very nice touch, Dennis). This is real "on the fly" runtime sorting.

If strings are supported, can variables be far off? A string, after all, is just a variable that doesn't change (at least, that's one way to look at it). If I can discover how multiple sort orders are handled by the templates, perhaps I can adapt the standard behaviors to the need to use variables.

To discern the standard behavior created by the templates, I opened an existing app and pressed the Source button from a four tabbed browse and went looking for known keys (i.e., sort orders). Big, big mistake, and Excedrin headache 82 (I'm still waiting for that flash of insight when OOP/ABC magically comes together). (Editor's note: A [set of classes](#) are available under [COSP](#) to create procedure call trees for ABC (and other) applications.)

To regain some semblance of control and simplify the generated code, I created a simple dictionary: one file, one AutoNumber key, three fields. Wizarding up a browse, I adjusted the defaults so that no locator would be used. Eliminating the locator also eliminates the method calls that setup the locator. This makes tracing the generated code ... less challenging.

The resulting (relevant) code in .INIT looks something like:

```
BRW1.AddSortOrder( ,CUS:CustomerNumberKey)
```

Not very informative is this? I already know that I have a key (sort order) on the customer number field. Instead of enlightenment, I find myself asking: Why is an existing key being added? What is it being added to?

Back to the drawing board: add a Tab with no key and enter an easily identified expression into Additional Sort Fields. Let's see what the templates do with this.

The resulting generated code is:

```
BRW1.AddSortOrder( , )
BRW1.AppendOrder( '-CUS:LastName' )
```

If the code for an existing key is unhelpful to examine, this code is at least an order of magnitude (or seven or eight) worse. This code seems to add a null sort order (an empty sort order, what is that?) and then adds something to it, somewhere (somehow).

Time to read the manual (you know, what you do when **all** else has failed and you're too confused to even frame much less ask an intelligent question on the newsgroups, or are too embarrassed).

The AddSortOrder method is a bit difficult to track down. If you search for "addsortorder" in the on-line help, you will see three entries: "AddSortOrder (add a sort order)," "AddSortOrder (specify a browse sort order)" and "AddSortOrder:BrowseClass ... ViewManagerClass." It is the last that documents the method the templates use in initializing a browse. Reading up on the other two will not hurt but will not explain the generated code (different number of parameters, different meanings).

Here is the core of the method specifications:

The AddSortOrder method specifies a sort order for the ViewManager object and returns a number identifying the sequence in which the sort order was added.

The AppendOrder method refines or extends the active sort order for the ViewManager object.

AppendOrder is fairly straightforward. It adds information to an existing sort order, specifically, the current order. So, if I append "CUS:CustomerNumber" (a unique value) to a key on CUS:LastName (not unique), I end up with an effective key of:

```
CUS:LastName , +CUS:CustomerNumber
```

which is a unique sort. Ok, that makes sense and I can understand what happened when I named Additional Sort Fields to a Tab in record order:

```
BRW1.AddSortOrder( , )
BRW1.AppendOrder( '-CUS:LastName' )
```

to get a descending sort on LastName.

Since `AddSortOrder` adds and sets the current order, I am adding "-CUS:LastName" to no key sorting information (record order). The result is the desired descending sort on LastName.

Also note from the documentation: `AppendOrder` can take a variable as its parameter. So, this approach is definitely headed in the right direction. That is, a string variable, containing the desired sort, will work in `AppendOrder` (in fact, what C5A does is enhance the template handling of Additional Sort Fields to accept either a string or a variable – using the standard " ' " and " ! " notation -- because the underlying `AppendOrder` method supports both).

But, I still have this one small question:

What are Sort Orders Added **to**?

This is the crux of the matter, the missing piece. This is what the `AddSortOrder` method does.

`AddSortOrder` is prototyped in `abbrowse.inc` and its code is in `abbrowse.clw`. You can trace the code there or I can summarize the important parts for you (or, at least, my interpretation of them).

When a browse is created, the browse manager also creates a queue, `BRWx.SORT` (don't try to access it, it is Private). This queue contains the information needed for the various sort orders, the developer's specifications for each Tab in the browse. The queue pointer is the sequence number returned by the `AddSortOrder` method referred to in the documentation quoted above.

As implemented, this data is added for Tabs 2 – n first. The data is taken from the prompts on the Conditional Behavior tab.

The procedure's default key, locator, filter, etc. are added to the queue last. This means that the current `SORT` queue entry is

```
?CurrentTab - 1
```

for all values of `CurrentTab` except 1.

This bit of intelligence makes sense of:

What Happens When Changing Tabs

The relevant part of the code generated for a Tab change (in `BRWx.ResetSort`) is:

```
IF CHOICE(?CurrentTab) = 2
  RETURN SELF.SetSort(1,Force)
ELSIF CHOICE(?CurrentTab) = 3
  RETURN SELF.SetSort(2,Force)
ELSIF CHOICE(?CurrentTab) = 4
  RETURN SELF.SetSort(3,Force)
ELSE
  RETURN SELF.SetSort(4,Force)
END
```

Notice the relation between the value of `CurrentTab` and the parameter passed to `SetSort`. It is  $n - 1$ , except for Tab 1, just as I stated.

What the `SetSort` method does is to use the first parameter as a pointer into the `SORT` queue and retrieves the indicated queue entry. Reset fields are reinitialized, the new order is applied and, if there is a filter, it is applied.

So, the entire implementation of multiple sort orders on Tabs turns on this queue. `AddSortOrder` and `AppendOrder` create/modify queue entries. `SetSort` retrieves and acts on those entries, provided you know the correct pointer. The queue pointer is calculated from `CurrentTab`. And that is why knowing the order in which entries are added to the queue is so important. (Perhaps this queue is `Private` precisely because it is central to the implementation of multiple sort orders. The `Private` attribute makes it much, much more difficult for the irresponsible to muck about with it.)

Simple.

## How the Templates Do Sort Orders

I think what has just been described is important enough to reiterate.

When Tabs are created, information is taken from the prompts on the Conditional Behavior tab. These include locator, key, filter and Additional Sort Fields information.

Information is collected starting with the second Tab (the first "conditional" behavior). The default locator, key, filter and Additional Sort Fields are added last. In this way, the default behavior is the target of an `ELSE` clause (makes sense, doesn't it?).

These data are added to a queue in the browse object. When the end user changes Tabs, the Tab number is used to calculate the pointer to retrieve this information from the queue.

## Next Time

There is rather a way to go yet. But I have covered enough material to keep your head busy for a while. Next time, I'll draw out the implications of this information and see if it provides a way to implement user defined sort orders.

[Download the example program](#)

---

[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.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).





# Clarion magazine

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

## The SQL Answer Cowboy

Andy "Cowboy" Stapleton is the acknowledged Clarion SQL guru and a regular presenter at Clarion conferences around the world. His company, [Cowboy Computing Solutions](#), produces SQL templates and classes for Clarion.

If you have an SQL question of a non-urgent nature for Andy, particularly if it reflects issues that will be of concern to other Clarion programmers, you can post it c/o Clarion Magazine as described below. Selected questions will be answered in the SQL Answer Cowboy column. (If you have an urgent SQL-related question, you're probably better advised to post it to the Drivers newsgroup on the [Topspeed news server](#).)

To submit your SQL question either email [advisor@clarionmag.com](mailto:advisor@clarionmag.com)

OR

fill out the following form and press the Submit button.

Your Name

Your Email Address

Your SQL Question:



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



Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## The Clarion Challenge

### Clarion Challenge Results

Edited by David Harms

Last month Clarion Magazine issued a [challenge to Clarion coders](#) to come up with the fastest, most compact code to determine how many billable periods of a given length of time existed between any start time and end time (which were not to be more than 24 hours apart). A number of you rose to the challenge, and the results brought up some interesting questions and observations.

I reduced all of the contributions to functions and placed them in a single exe for side-by side testing. You can download the [source with project file](#) or the [source and a locally compiled 32 bit exe](#).

Figure 1 shows a screen shot of the test program. I subjected the contributed functions to a fairly small number of tests to check basic compliance (your results may vary – I used a Dell Inspiron 3000 notebook). Figure 1 shows the first (default) test which checks for the correct calculation of fifteen minute non-rounded periods.

**BKO**  
Enterprises, Inc.

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

etc  
2000

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

Figure 1. Fifteen minute blocks, non-rounded.

Name	Units	Elapsed
David Bayliss	4	2.92
Peter Gysegem A	3	2.97
Brian Staff	4	3.57
Gordon Smith	4	3.73
Nik Johnson A	4	6.04
Alan Telford	4	6.26
Carl Barnes	4	6.26
John Cunningham	4	7.08
Poul Jensen	4	7.08
Kurt Pawlikowski	3	7.31
Jeff Starve	4	8.24
Nik Johnson B	4	11.59

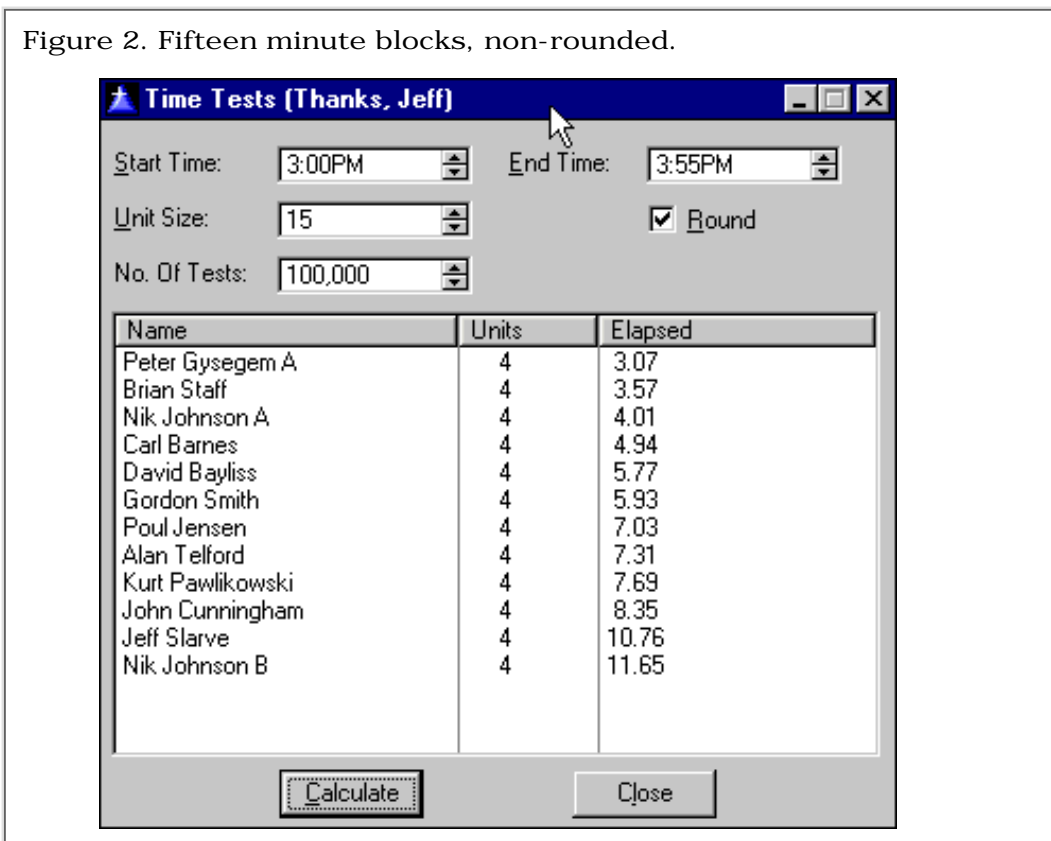
All but two of the methods showed the correct value. With the rounding box checked, all methods showed the correct value.

In order to get reasonably accurate time samples I used 100,000 iterations per function (keep in mind that the timer clock resolution is only 1/18 second). A faster testing solution would be to use API functions to get smaller timer ticks. Maybe next time. There are also some questions about whether the test program introduces variances into the results, so there may be some slight inaccuracy in the results (second place finishers, take note!).

Not surprisingly, David Bayliss clocked the fastest time in the first test. But then it's his compiler. Among the correct answers, Brian Staff came in second. Several of the functions has problems with the non-rounded calculation.

When the rounding flag was set, Mr. Bayliss fell to fifth place, and Peter Gysegem took top honors, as shown in Figure 2.

Figure 2. Fifteen minute blocks, non-rounded.



When testing code it's always important to check boundary conditions. With the start time set to 3:00 p.m. and the end time to 4:00 p.m., two functions returned five rather than four units. Although the requirements are open to interpretation, I think few users would expect one hour to show as five billable fifteen minute periods.

Gordon Smith provided the following test conditions which almost no one correctly handled in all cases (expected results are shown in parentheses).

Start time=3.00pm, end time = 2.59pm, unit size= 1,  
round=off. (1439)

Start time= 12.00am, eend time = 12.00pm, unit size= 1,  
round=off. (720)

Start time 11:59pm, end time = 12:00am, unit size=3,  
round=off. (1)

Start time=3.00pm, end time = 3.00pm, unit size= 1,  
round=off. (1440)

Start time=3.00pm, end time = 3.00pm, unit size=60,  
round=off. (24)

Starting at 3:00 p.m. and ending at 2:30 p.m. with rounding on yielded varying results (23 or 24 hours). Which is correct, and which should be correct? This is a less-clear situation than the number of units in an hour, so no error was assigned, except to the specification writer. This underlies the importance of anticipating such situations in design as well as in testing.

A composite rating of function speeds is as follows:

\* Failed one or more tests

Peter Gysegem (A) *	6.04
Brian Staff	7.14
David Bayliss *	8.69
Gordon Smith *	9.66
Nik Johnson (A)	10.05
Carl Barnes	11.2
Alan Telford *	13.57
Poul Jensen *	14.11
Kurt Pawlikoski *	15
John Cunningham *	15.43
Jeff Slarve *	19
Nik Johnson (B) *	23.24

As you can see, only Brian Staff and Carl Barnes survived all the tests (at least so far) with Brian having the fastest time.

It's a bit more difficult to rate the size of the functions since not all contributors used the smallest possible labels and minimum spacing, and not all code was received in the form of a function. [Click here](#) to view the program source code and make your own assessment.

Contributor comments: Peter Gysegem

I was surprised to find that passing the variables as a named group didn't change the speed within the margins of error in my tests. I also tried a couple different methods for calculating the value of *Elapsed* with no statistically significant difference in times:

```
Elapsed = (EndTime - StartTime) + 8640000 * (EndDate - StartDate)

IF BXOR(StartDate, EndDate)
  Elapsed = EndTime + 8640000 - StartTime
ELSE
  Elapsed = EndTime - StartTime
END
```

My results show that the most readable (and maintainable) code is also the fastest and that inline nested functions slow things down while often making the code harder to maintain.

Contributor comments: Carl Barnes

I was amazed at how fast DAB's code was but yet it was very simple and straightforward demonstrating his past statements that "straightforward" is the way you should write your code and let the compiler writer optimize it.

I would say the *ROUND* function in Clarion is very efficient, I tried to not use any function thinking it would be faster. Possibly my code is too nested to optimize where simple straight forward code optimizes the best. I would guess I did not pay attention to type conversions I am causing.

### Summary

Although this seemed like a fairly straightforward challenge, the resulting code (as is often the case) demonstrated that there are a number of facets to the design that were not adequately considered in the original specification. The fact that most of the functions failed one or more tests is in part a reflection of the lack of detail in the specification.

Future editions of the Clarion Challenge will probably include specific tests and will focus less on code size and more on speed and maintainability.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## Feature Article

### Understanding FieldClass

or

#### How To Put Generated Code On A Weight Loss Plan

by David Bayliss

When I sent around the initial design proposal for what later became the ABC system one of the claimed benefits was a code reduction in user procedures of around half to two thirds. At the time this was viewed with some skepticism and so we set 30 percent as a reasonable goal. In the end we actually achieved around 92-94 percent, and the field class was chiefly responsible for the extra 30 percent.

To appreciate why, you need to consider how certain parts of the ABC system would be coded if the FieldClass didn't exist. For the sake of concreteness I am going to use the example I used many moons ago when I was trying to persuade Tom Moseley that OOP could work in the templates.

#### Example 1 : Updating a link field

One constant bugbear in CW2 was overflow of the appname\_ru (referential integrity, or RI) module (to >64K) on any sizeable or complex dictionary. One aim of OOP was to reduce this problem. Although there were other procedures the heart of the referential integrity can be shown by pseudo-code for the RI update function.

Listing 1 assumes F1 & F2 are the related files (on the keys F1:K & F2:K with 2 & 3 components respectively, KC1, KC2, KC3 etc).

Listing 1. Code to cascade RI updates.

```
CLEAR(F2:KC3,-1) ! Clear minor-most components
F2:KC2 = F1:KC2
F2:KC1 = F1:KC1
SET(F2:K,F2:K)
LOOP
  NEXT(F2)
  IF F2:KC1 <> F1:KC1 OR F2:KC2 <> F1:KC2 THEN
    BREAK ! No longer meeting range
  END
  F2:KC2 = New:F1:KC2
  F2:KC1 = New:F1:KC1
  Cascade_Updates
END
```

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

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



I have left out several vital details but this is enough to show the nature of the problem. This code fragment (and every bit of file IO/error handling that goes with it) appears for every relation with RI restrictions on it. If you have 100 files, 250 relations means 250 copies of the code. It's not too surprising RI code frequently blows the segment limits in the legacy templates.

The challenge is to "proceduralize" the above code.

When trying to abstract out an algorithm I like to go through the code colouring the lines. I use three colours: blue for base classes; green for parameterized base classes; black for instance specific code. In my first pass over the code I just pick out the blue stuff: those lines of code which will always be the same no-matter which of the 250 copies of the code I am looking at.

Listing 2. Code to cascade RI updates with common code in blue.

```
CLEAR(F2:KC3,-1) ! Clear minor-most components
F2:KC2 = F1:KC2
F2:KC1 = F1:KC1
SET(F2:K,F2:K)
LOOP
  NEXT(F2)
  IF F2:KC1 <> F1:KC1 OR F2:KC2 <> F1:KC2 THEN
    BREAK ! No longer meeting range
  END
  F2:KC2 = New:F1:KC2
  F2:KC1 = New:F1:KC1
  Cascade_Updates
END
```

This ranks as grim. The vast bulk of the code is actually different between the 250 copies. You could put the loop in the base class and call out for the header and loop body, but the total lines of code (once you've allowed for two new procedures) actually goes up. Listing 3 shows five lines of code that replace the three blue lines.

Listing 3. Method to call RI virtuals.

```
RelationClass.UpdateSecondary PROCEDURE
CODE
SELF.UpdateSecondaryInit ! Virtual call- override for every relation
LOOP
  IF SELF.UpdateSecondaryIterate THEN BREAK .
END
```

Perhaps the green pen will yield better results. This time I can colour lines with variables provided I then add a parameter to the procedure prototype to allow the value to be substituted.

Listing 4. Parameterized and base class code in green.

```
CLEAR(F2:KC3,-1) ! Clear minor-most components
F2:KC2 = F1:KC2
F2:KC1 = F1:KC1
SET(F2:K,F2:K)
LOOP
  NEXT(F2)
  IF F2:KC1 <> F1:KC1 OR F2:KC2 <> F1:KC2 THEN
    BREAK ! No longer meeting range
  END
  F2:KC2 = New:F1:KC2
  F2:KC1 = New:F1:KC1
  Cascade_Updates
END
```

There is a subtlety here. Why didn't I colour the CLEAR? Because the number of components to be cleared is not a function of the relation, it is a function of the key used by the secondary file. Thus you cannot readily parameterize it. So now you get the code shown in Listing 5.

Listing 5. Parameterized method to handle RI update.

```
RelationClass.UpdateSecondary PROCEDURE(File F, Key K, *? F1Field1,*? F1Field2, *?
F2Field1, *?F2Field2, ? New1, ?New2)
CODE
SELF.UpdateSecondaryClear ! Virtual call- override for every relation
F2Field1 = F1Field1
F2Field2 = F1Field2
LOOP
NEXT(F)
IF F2Field1 <> F1Field1 OR F2Field2<>F1Field1 THEN
BREAK
END
F2Field2 = New2
F2Field1 = New1
SELF.Cascade ! Virtual
END
```

Now each of the 250 code lumps becomes two small virtual procedures and one base class call (with 8 parameters). This cuts down the line count although the actual amount of code generated is still quite high. Each \*? Parameters costs 30+ bytes, so 6 of them is 180 bytes. I have also snuck a little bug past you. I have been assuming throughout that there are two linking fields. There can (of course) be 1, or 3 or 4 etc. So you need copies of the UpdateSecondary procedure (and the other four) for each possible number of pairs of fields.

Now I have greatly shrunk the code (i.e. there will be no more 64K problems) and just about everything has been abstracted. Every now and then someone will call to complain that ABC doesn't support 9 linking fields in a relation and we can simply write a new UpdateSecondary9 (with 27 \*? Parameters at 600+ bytes per call).

But code abstraction doesn't have to end here. This design has UpdateSecondary1, UpdateSecondary2 etc and these procedures are really the same except in the number of parameters passed in. You can write a generalized UpdateSecondary procedure (except it won't compile!) as shown in Listing 6.

Listing 6. A general UpdateSecondary procedure.

```
RelationClass.UpdateSecondaryN PROCEDURE(File F, Key K, (*? F1Field1, *? F2Field1, ?
New1) * N)
CODE
SELF.UpdateSecondaryClear ! Virtual call- override for every relation
LOOP N Times
F2FieldN = F1FieldN
END
LOOP
NEXT(F)
LOOP N Times
IF F2FieldN <> F1FieldN THEN BREAK OuterLoop.
END
LOOP N Times
F2FieldN = NewN
END
SELF.Cascade ! Virtual
END
```

Listing 6 won't compile because that isn't a legal procedure prototype. But you can see the idea – I want to be able to pass in any number of fields without having to define the fields ahead of time.

## Example 2 : Formatting a browse line

Another place that presented problems was the browse code. Most of the engine can disappear into a procedure (Bruce actually did this for CDD3.0). However there are three very large routines you cannot take down: filling a browse queue from data; filling the record buffer from the browse; and seeing if any data in the browse queue has changed. These three routines essentially look like this :

```
BrowseQ:Field1 = File:Field1  
BrowseQ:Field2 = File:Field2  
BrowseQ:Field3 = OtherFile:Field7
```

where "fill buffer" goes the opposite way to "reset buffer."

Easy you say, that's the same as parameterizing. But think about it! Restricting the number of linking fields to 9 is one thing, but the number of browse columns? We would have to go up to 100 just to avoid getting shot by the alpha testers! On the other hand if only I could get the `LOOP N Times` code from Listing 6 to compile then this really would be so easy.

(Some of you may think we could use the `:=` syntax to move across the corresponding fields. In general that doesn't work because it doesn't allow for browse columns defined by local variables. It also suffers if you have two files in the browse with clashing field names).

In my opinion it was this problem that killed the CDD browse engine. Because the engine had to call back to the main code so frequently to do almost anything (and they didn't have the virtual mechanism to clean things up) the code became almost impenetrable. So the engine died, the inline browse appeared, and the browse procedure became our main bugbear for over five years.

## What's The Real Requirement?

The job then becomes one of defining what it actually is about the `LOOP N Times` code that will solve the problem. I think it comes down to the following :

I need to be able to pass around a list of one or more field pairs which can then be manipulated as a single entity.

Think about those last two words; they are the key. If I can embody the `LOOP N Times` into a single line of code then I have the problem cracked.

My expression field pairs also betrays another consideration. In the browse case there are only ever two fields that are really interesting; for the RI code there are three interesting values (child fields, parent fields, new parent fields). The prototype for the `UpdateSecondary` is also interesting. Note that the fields pertaining to the files are prototyped as `*?`, meaning they can be assigned to and from. The new fields are only ever used by value. It turns out that (in this example, at least) there are typically 4 different cases :-

1. Single field. This is a list of fields with no partner. In fact the components of a key are stored this way which makes it possible to bring the `CLEAR(keycomponent)` into the base class as well!
2. Single field – buffered. These are fields which have to have a snapshot of their values taken without changing and 'real' program variables so the variables can be later compared to those values.
3. Two fields. Two sets of fields, either of which can be assigned to and from the other.
4. Two fields – buffered. This is the most complex case of two sets of fields where either one may need snap-shotting.

Because the fourth case is much heavier than the others (although related) we decided to assign it to its own class which is derived from the field pairs class.

## The Implementation - Any Ideas?

In order to understand how this class works you certainly need to understand queues but you also need to understand the ANY datatype. This is given an excellent coverage in the manuals which I shan't repeat. However, the key here is this: an ANY can act like a \*? parameter OR a ? parameter dependant upon how you assign to it.

Specifically, if an ANY variable is NULL (has no value) then a straight value assignment to it produces a value ANY, while a reference assignment to it produces a variant any. Listing 7 shows an example.

Listing 7. Using ANYs to store values and references.

```
MyAny &= NULL
Field = 22
MyAny = Field           ! MyAny = 22
Field = 42               ! MyAny = 22
MyAny = 50               ! Field = 42, MyAny = 50
MyAny &= Field           ! MyAny = 42
Field = 62               ! MyAny = 62
MyAny = 72               ! Field = 72
```

Warning: CLEAR(MyAny) is equivalent to MyAny = 0. It is NOT the same as MyAny &= NULL.

```
FieldPairsClass.Init PROCEDURE
```

The Init procedure is simple enough to use. It creates the queue that forms the basis of the class. A slight oddity is the call to Kill first. This is to allow a FieldPairsClass to be used and reused within a procedure. (Effectively Init acts as a Reset.)

```
FieldPairsClass.AddItem PROCEDURE(*? Left)
```

There are two notional AddItem methods (the second called AddPair). This one is used for cases 1 & 2. Note the ASSERT to insure Init has been called. The CLEAR is dealing with some (rather nasty) memory management issues when dealing with ANY in queues (see the manual). The incoming variable is &= into the left hand queue element. It is then = into the right hand element. This distinction is crucial ([see above](#)). It means that simply Adding a field is enough to snap-shot it so that it can be reset (or tested for difference) at a later stage. The parameter is called Left because you can think of it as something you can assign into (and which therefore appears on the left hand side of an assignment (=) operator.

```
FieldPairsClass.AddPair PROCEDURE(*? Left,*? Right)
```

This method is used for variant 3. Other comments are the same as AddItem. Note also that in this case left & right do not have any real significance. it is just a non-suggestive way of labeling the two entities.

```
FieldPairsClass.AssignLeftToRight PROCEDURE
```

This procedure is really meaningless in variant one (actually it converts a variant one into a variant two). In variant 2 this can be seen as a way of snapshotting the current values of all the variables. In variant 3 all the values from the variables passed in as 'lefts' will be copied into the variables passed in as 'rights'.

Warning: Note the PUT after the assignment. This is because an assignment to an ANY variable can actually change the memory block allocated to the ANY. Hence you have to store the queue after an assignment even if you know the ANY is a variant ANY.

```
FieldPairsClass.AssignRightToLeft PROCEDURE
```

Again the use of this suggests you are not really in the variant 1 case. In variant 2 it has the effect of restoring all the variables passed in as Lefts to the values they had when an AssignLeftToRight was last done. (Which could be the implicit one at the Additem point). In variant three this is an assignment from the variables passed as Rights to the variables passed as Lefts.

```
FieldPairsClass.ClearLeft PROCEDURE
```

This has the same effect for all three variants, it **CLEARs** the variables passed in as Lefts. This is not the same as assigning to zero, because the left-hand side could be a string. It is also not quite the same as assigning to a blank string (consider Cstrings & Pstrings). Now you could argue that it is the same as assigning to a zero length string, which is true, but only by coincidence. This illustrates one of the big pitfalls of having a language "guru" doing low-level classes. You can use your low-level knowledge to build assumptions into the system that are not required. The fact that presently all Clarion data-types can be **CLEARed** by assigning a zero length string is a very dangerous fact to build into a set of base classes (consider what would happen if you could pass a mixed-type group as a \*?). The clearing mechanism is there to protect you from such assumptions, so the base classes use the full language facility where they can.

Note further that CLEAR(SELF.List.Left) is very different from SELF.List.Left &= NULL ([see above](#)).

```
FieldPairsClass.ClearRight PROCEDURE
```

In variant 2 this clears the buffer values, in variant 3 it clears the variables passed in as Rights. This method is subject to the same considerations as ClearLeft.

```
FieldPairsClass.EqualLeftRight PROCEDURE
```

In variant 2 this compares the current values in each of the Lefts against the last snap-shotted values. It returns a zero if any of the values differ. In variant 3 it compares each Left-Right passed in and returns a zero if there are any differences.

Note that this procedure effectively does a short-circuit evaluation which means the function returns as soon as a deviation is found. It demonstrates one of the reasons that I believe certain programming mantras can and should be violated in a controlled environment.

First the controlled environment. EqualLeftRight is 10 lines long, it fits on one screen and (I claim) should be understandable in one bite by a half-way competent programmer.

Now for the mantra. Good structured programming will teach you that any given procedure should have precisely one entry point and precisely one exit point. This procedure has two exit points. Why? Certainly efficiency, and also (I claim) clarity. Consider the obvious alternative in Listing 8.

Listing 8. A single exit point alternative to EqualLeftRight.

```
FieldPairsClass.EqualLeftRight PROCEDURE
I UNSIGNED,AUTO
B BYTE(1)
CODE
LOOP I = 1 TO RECORDS(SELF.List)
  GET(SELF.List,I)
  IF SELF.List.Left <> SELF.List.Right
    B = 0
  END
END
RETURN B
```

Now the method has the required one exit point. However there is an extra line of code and there are two extra assignments (`BYTE(1)` is an implicit assignment). But the real pain is more subtle. Imagine a big field list (100 fields) in which you are checking for a difference (say after an Edit-In-Place operation on a browse). This code will check all 100 fields even if the first one sets `B` to zero!

So you end up having to put a `BREAK` into the `IF` condition or code an `UNTIL` at the tail of the `LOOP`. The latter is less efficient still. The former is efficient but if you now draw a flow diagram of your algorithm you will find exactly the same logical structure as coding a `RETURN` but it took you 20% longer to say it!

This brings me to the Bayliss mantra: keep it short and to the point, but then don't compromise!

```
FieldPairsClass.Equal PROCEDURE
```

This is simply a logical short-hand for people using the `FieldPairsClass` as opposed to the `BufferedPairsClass` (where the explicit `LeftRight` is helpful).

```
FieldPairsClass.Kill PROCEDURE
```

Check over this code. The destruction sequence of queues with `ANYs` needs careful work. First you have to null out all of the any variables, then you can dispose of the list.

## Derived Classes

```
BufferedFieldClass
```

This class is really just an extension to the `FieldClass` to handle case 4. Two fields are paired and there is a shadow third value. In some ways this makes it easier to understand than the `FieldClass`. If ever `Left` or `Right` are assigned to/from then it is the values in the underlying fields that are being used. Buffer means the shadow which never effects any values in the "real" program.

## Queue Derivation

The `BufferedFieldClass` is derived from the `FieldClass`; that is to say whenever a buffered field class is being used without reference to the shadow value you can simply call the same functions as you would for a case 3 of the field class. The buffered field class is an extension for the case when buffering is needed. Now we could simply have implemented the `BufferedFieldClass` and used it for cases 1 through 3. The main reason we didn't is one of efficiency. `ANY` variables work extremely slowly compared to standard Clarion variables (about 30x slower, or similar to Visual Basic) and therefore maintaining an extra 1 or 2 for the very common cases (1 through 3) was deemed unwise. The separation also enables the field class to have a relatively small, clean interface.



```
BufferedPairsClass.Init PROCEDURE
```

This procedure demonstrates a simple problem, with a simple enough fix, but to the unwary it can be very confusing. The `FieldClass` contains a reference to a `FieldPairs` queue (with `Left` & `Right` ANYs). This is NEW/DISPOSED in the `FieldClass` `Init` and `Kill` methods. The `BufferedFieldPairs` class has a reference to a buffered queue with three fields. Now here is the problem: if the `FieldClass` and `BufferedFieldClass` both have `Init` and `Kill` called then there will be two separate queues pointed to by two separate references. So the `BufferedFieldClass` `Init` method does not call its parent. As a result there is only one copy of the queue.

But there is a subtler problem. Suppose the `Equal` method is called. This drills down to `FieldPairs.EqualLeftRight` which expects the `SELF.List` reference to be filled in, which it won't be. Bang!

Here is the fix. The `BufferedPairsQueue` is (very deliberately) just the `FieldPairsQueue` with extra fields added. The `Init` method =& the `List` in the `FieldPairsClass` to the `RealList` in the derived class. Now the methods in `FieldClass` can access the same queue as those of the `BufferedFieldClass` but via a different reference.

Tech note: A particularly nice feature of queue and class references is that they contain type information. Thus `CLEAR(MyQueueReference)` will always clear the whole queue buffer. Similarly `ADD(Queue)` works on the whole queue.

```
BufferedPairsClass.AddPair PROCEDURE(*? Left,*? Right)
```

This method overrides the equivalent method in the base class. Later versions of it actually contain some rather intricate code to fix a subtle bug that I missed on the first lap. All the code is really trying to do is reference assign `Left` and `Right` (as per the parent function) and then `CLEAR` the buffer value (because I don't know whether to assign it to `Left` or `Right`). But the question becomes, what does it mean to clear an any variable? (See discussion on [ClearLeft](#).) What I really want is to assign it to a value which will compare equal to the `Left` or `Right` variables if they have been cleared. The only general way I could think of doing this was to clear the `Right` variable and then assign it to the buffer. Of course people might object to me doing that so I temp-store it first.

I think the other methods are fairly self-explanatory given the `FieldClass` explanations.

Finally

ANY variables (and type polymorphism) are key strengths of the Clarion language that make it possible to code complex database algorithms in a totally generic and safe way. The two field classes extend this paradigm up to lists of field pairs. If you scan the ABC sources you will find the field pairs classes are intrinsic to files, browses, drop combos and edit-in-place. If you scan generated source you will find `AddPairs` popping up very frequently. The combined effect of these facts is that most procedures can be generated without any need to derive the browse or file objects. This simplifies and reduces the amount of code required to use these classes and gives Clarion an implementation edge (from template or hand-code) over C++, VB and Object Pascal.

I hope this `FieldClass` design overview has given you an insight to one of the fundamental building blocks of the ABC system.

---

[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

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

## Product Review

### The Clarion Class Browser

#### Freeware For The OO Programmer

Reviewed by Dave Harms

Shortly after TopSpeed released the first ABC templates Gordon Smith (aka "Le Schmoo") produced the Clarion Class Browser. Class browsers are a common feature in OO languages and are a much better way to view class source than viewing the source files directly.

The Clarion Application Generator does come with a class browser of sorts already, it's true. But this built-in browser has limited capabilities and cannot be run outside the AppGen. For most situations, and definitely when you're working with non-ABC classes, Le Schmoo's browser is the tool of choice. And the price is certainly right: it's free.

The class browser comes as a single executable setup program and installs into the directory you specify. This is a locally-compiled EXE – you do not need a particular runtime DLL.

After you install and run the program you'll need to create at least one class library file, which is a database of class header files. Choose File|New and provide a location and name for the database (the file will have the extension .clb). Click OK, and you'll see the library properties window, shown in Figure 1.

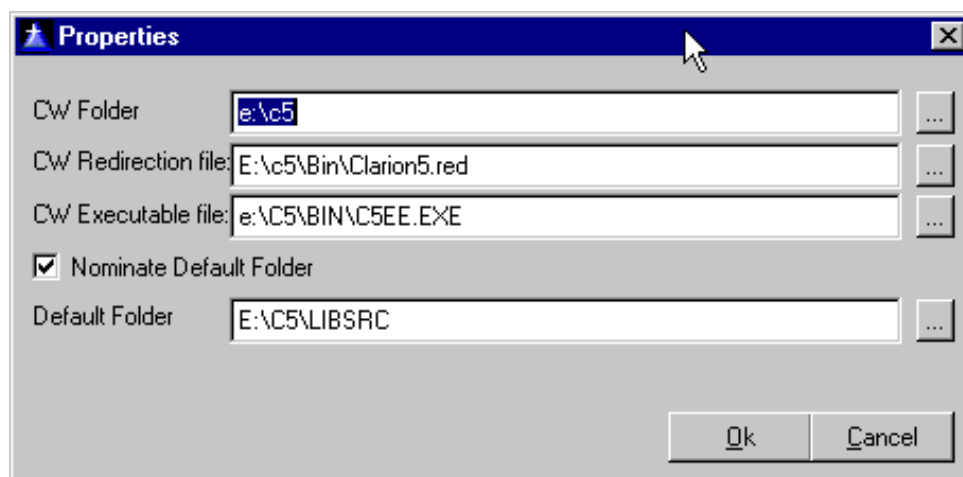
**BKO**  
Enterprises, Inc.

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

etc  
2000

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

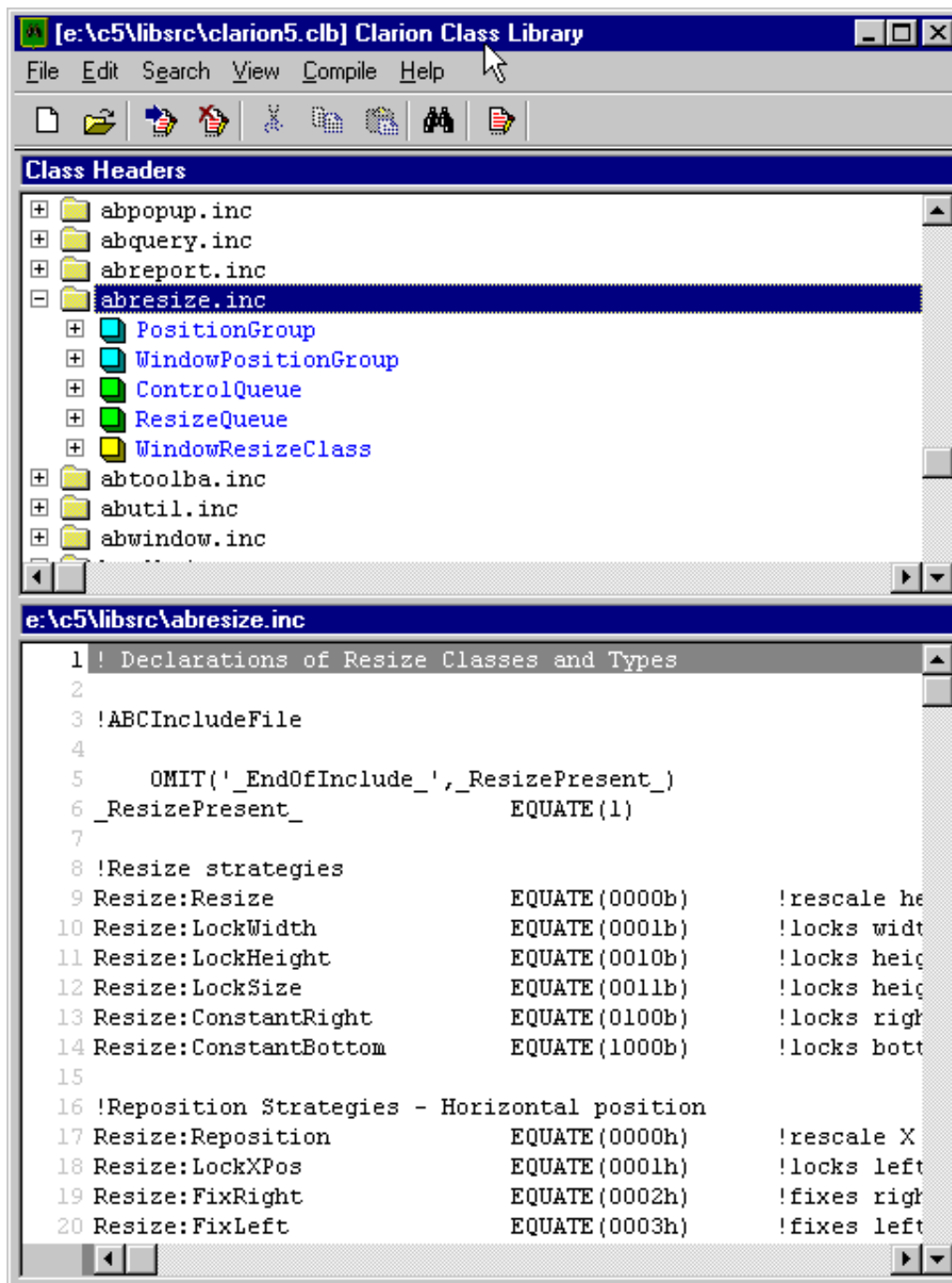
Figure 1. The class libraries properties window.



I always dutifully fill in the fields as requested, but I have noticed that no matter what I specify for the default folder, the browser always looks in its own directory first when I wish to add a header. This may be because I have the class browser installed on a different drive than Clarion. (You can always go back and change these settings by choosing File|Properties.)

This directory amnesia isn't a big problem as you can add multiple classes at once, thereby avoiding repeated trips to the file dialog. Choose Edit|Insert OOP Header (or press Ins) to bring up a file dialog that looks for files ending in .INC. You select multiple files using the shift or control keys along with the mouse (or cursor keys). Click OK to add the header (.INC) files, which will provide the browser with the class declarations. Figure 2 shows the browser with the ABC (and several other) class headers loaded.

Figure 2. The Class Browser main window.

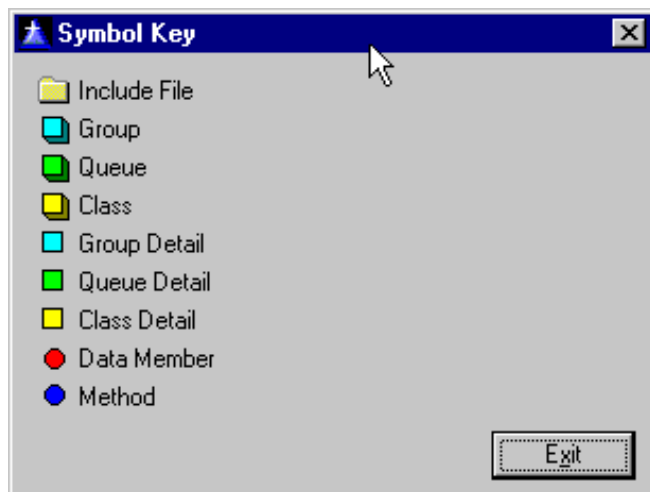


As Figure 2 shows, the top window shows the headers and their contents in a list box, and the bottom window shows the source code. I'll refer to these windows as the header and source windows. You can also have the source window on the right hand side if you prefer. This is how I work, but because of limitations on image size in this document most of the screen shots show the source on the bottom. [Click here](#) for a large image of the side by side view.

If you select a class element in the header window that is declared in the INC file (such as a group or queue, or the first line of a class declaration) the corresponding line is located and highlighted in the source window. If you select a method in the header window, the method source (from the corresponding CLW file) is shown.

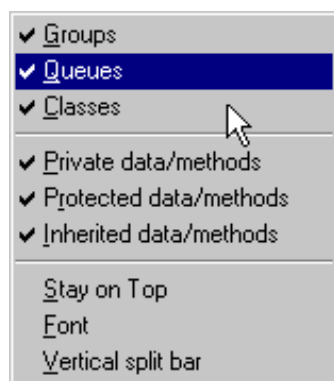
As Figure 2 shows, header elements are identified by symbols. Figure 3 shows the symbol key (available from the Help menu).

Figure 3. The symbol key.



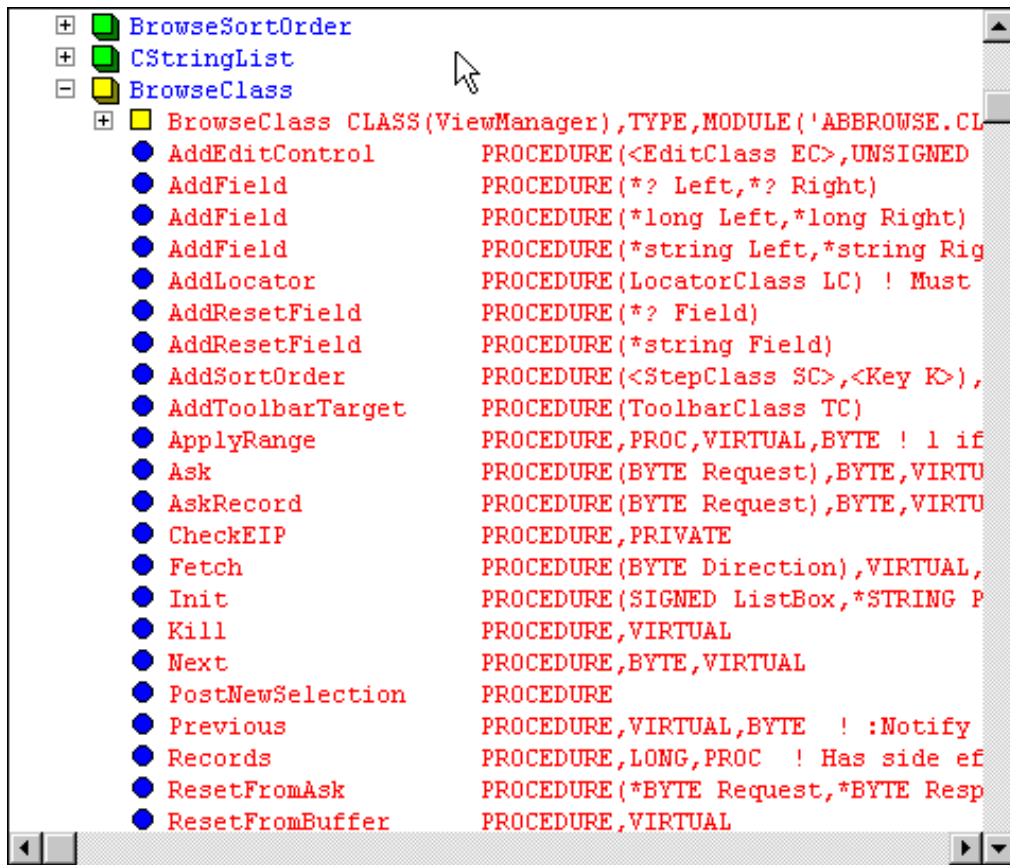
You can decide which of the header elements to display by making selections from the View menu, shown in Figure 4.

Figure 4. The view menu.



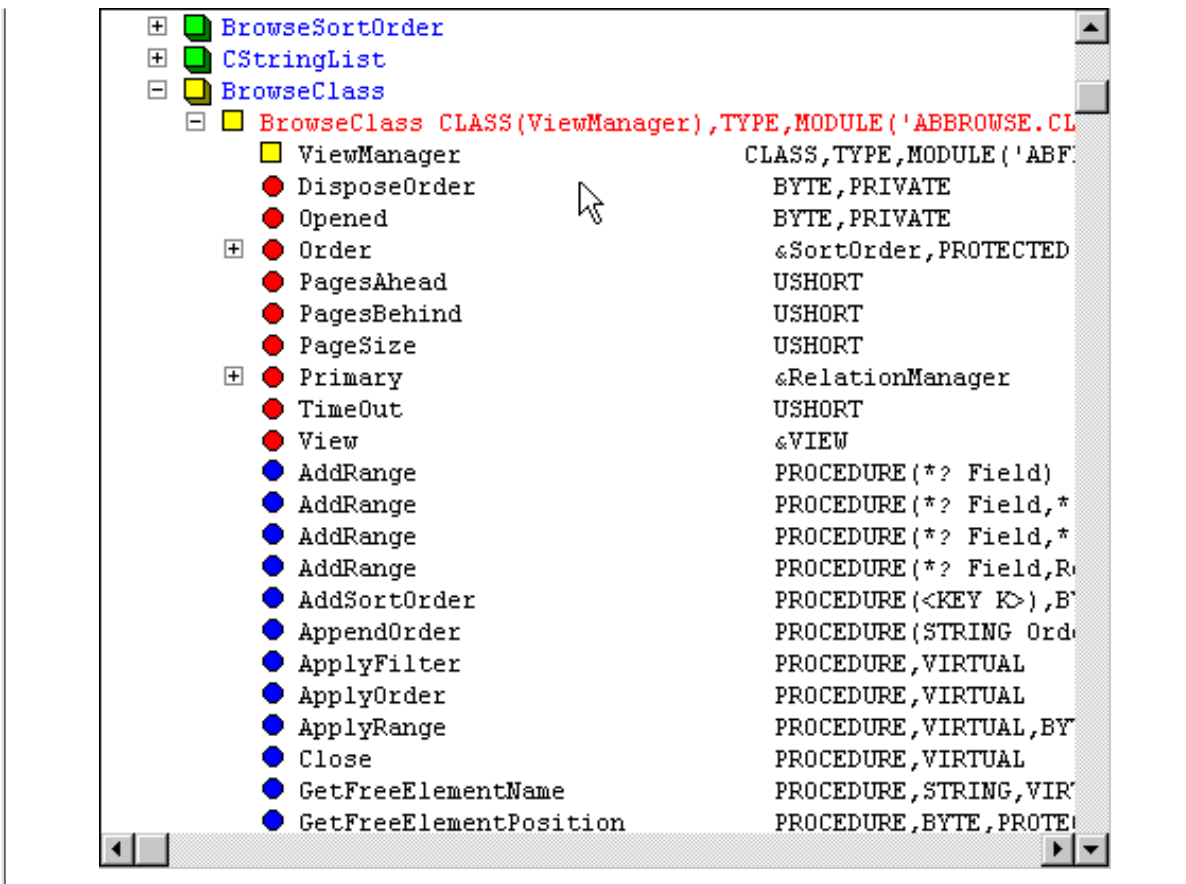
One of the key benefits of a class browser is it shows you the whole structure of derived classes. Figure 5 shows the beginning of the ABC BrowseClass, which is derived from ViewManager.

Figure 5. The ABC BrowseClass.



The + sign before the class declaration indicates that the list can be exploded at that point to show ViewManager's declaration, as shown in Figure 6.

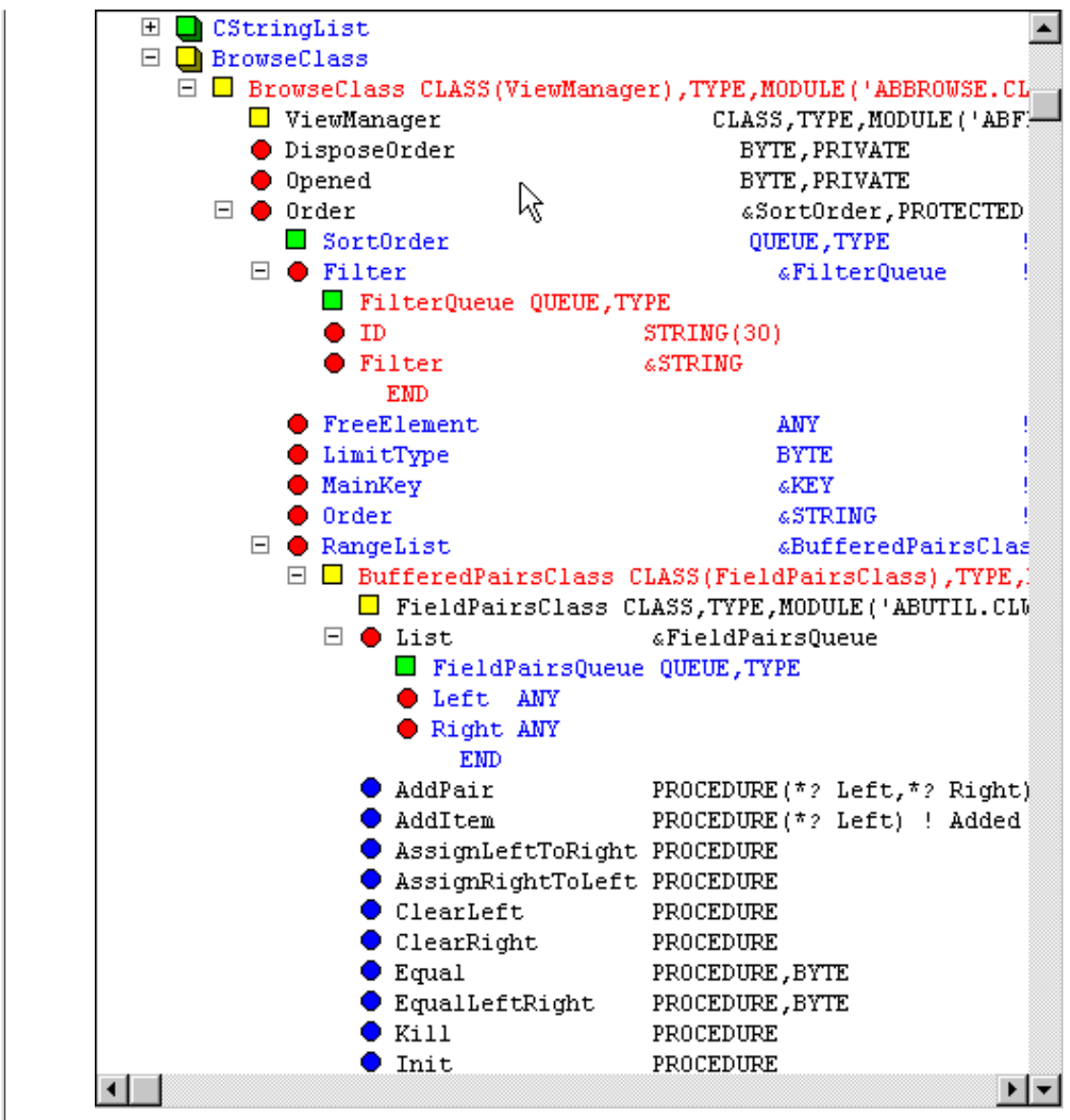
Figure 6. The ABC BrowseClass exploded to show the parent ViewManager class.



Note that within ViewManager there are other declarations (Order and Primary) which can also be expanded. Order can be further expanded to show two of its complex elements: Filter (another queue) and RangeList. RangeList of type BufferedFieldPairsClass which is derived from FieldPairsClass, and FieldPairsClass contains a queue. Figure 7 shows these declarations.

Figure 7. Deep in the bowels of BrowseClass.

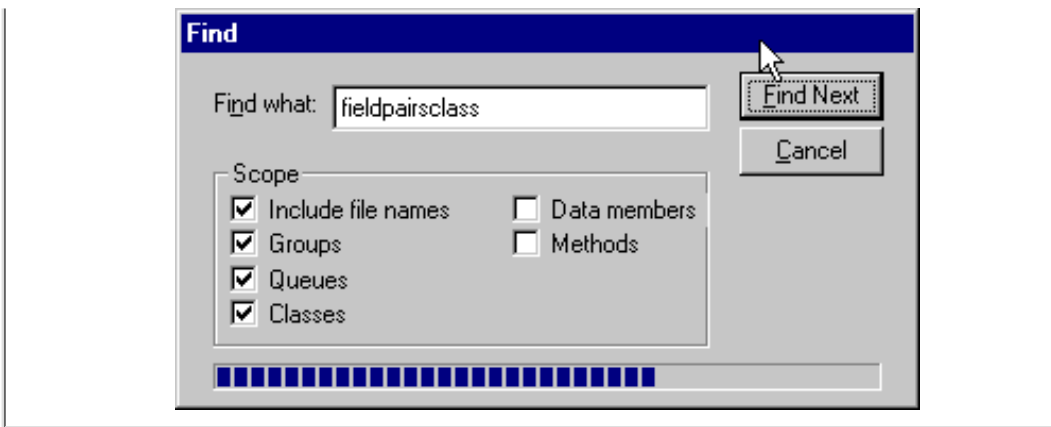




Without a class browser, the only way you could get a picture of the actual structure of BrowseClass would be to go digging through the header files (which is exactly what the class browser does for you), and then you'd have to go through the source files to see what the methods do. With the class browser, the method source is also only a click away.

The class browser also has a Find feature which lets you search all or selected parts of the class library. Once you've found the first instance you may wish to close the modal Find window and use the Edit|Find Next feature (or just press F3).

Figure 8. The Find window.



The class browser also has a Compile menu which has but one item: Generate Project. This option will create a PRJ and corresponding EXP (export) file (with the same name as the class library) and a set of modified INC files for the classes you have loaded. This is NOT intended for use with the ABC classes since there are also typically global variables and files involved when setting up multi-dll applications, but it's a nifty solution for creating handcoded dlls as it handles the export file creation. The EXP created is for 16 bit applications. You can use it for 32 bit apps but be aware that the SIGNED and UNSIGNED data types mangle differently in 32 bits so if you're using these as parameters you'll have to modify the EXP file.

One item on the toolbar which isn't on the menu anywhere is the Edit Source button. Use this button to switch to edit mode in the source menu, and to switch back (and save your changes). Although this is a nice feature, I think I'd prefer to use the Clarion editor to work with my source.






## Documentation





One drawback to the class browser is the complete lack of documentation. Fortunately, most features (except perhaps DLL creation) are fairly intuitive. Another weakness is that the Class Browser is time-limited, and several times per year you will need to download an updated version from Gordon's web site. As I use the class browser heavily it seems I'm occasionally the first person to discover the browser has expired, but I've always been able to obtain an updated version in short order, most recently the same day.

I've given the the Class Browser a "three pyramid" rating rather than a four only because there is no documentation explaining where to get support. As this is a mature product you most likely won't need any, but the author's email address is in Help|About. Earlier in the product's life I did report some bugs to Gordon and always received prompt action.

If you're interested in exploring the inner workings of ABC, I suggest you use the Class Browser in conjunction with cciProfilerClass to generate a method call tree. As you follow down the log of method calls use the browser to locate the source and see what code is being executed.

The Class Browser is one of my most-used utilities and I recommend it highly. You can download it from <http://www.iol.ie/~schmoo>. Also available at this site: Compile Manager, a tool for automated compiles of multi-app projects.

PRODUCT RATING	
Documentation	
Features	
Support	
Value	
Overall	

Legend	
Excellent	
Good	
Fair	
Poor	

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## News Feature

### Technology Preview: Clarion Does COM!

by Ross Santos

Many of us in the Clarion for Windows world have clamored for tighter integration of OLE components (now known as COM, which stands for Component Object Model) in Clarion for Windows. Well, TopSpeed Corp. has listened, and busily, behind closed doors at their UK Development Centre, they have been hard at work developing the best solution for us all: Clarion COM. That's right! Clarion for Windows will finally do COM.

We've got templates, we've got objects and soon we'll have COM in all its glory, and with dot syntax too! What more could we ask for!

The new Clarion COM layer will support both IDispatch (VB style) as well as IUnknown (C++) style COM models. Clarion for Windows programs will be COMable too. That means that you can create a program that will support COM interfaces and functionality, breaking the barrier between disparate languages.

Furthermore, a sophisticated COM Object class will handle many of the dirty details, making programming COM much simpler and easier to master than in other well known languages (nudge nudge wink). As well, BSTRs and VARIANTs will be integrated into the Clarion language. So keep your eyes peeled for the first of a series of articles written by me (Ross A. Santos) on the new Clarion COM right here at ClarionMag.com.

---

## About COM

Object-oriented programming is a great thing, but it's even better when you can use objects from another source without having to link them into your application. In the Windows world this is mainly accomplished through COM, Microsoft's Component Object Model (for non-Windows or multiplatform work [CORBA](#), a competing standard to Microsoft's Distributed COM, is often used). COM is the backbone of OLE, and essentially is an Object Request Broker (ORB) which manages messages between objects which conform to the COM standard.

### Links

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

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

Microsoft's COM Home Page

<http://www.microsoft.com/com/default.asp>

Overview of COM technologies

<http://www.microsoft.com/com/comTech.asp>

Microsoft COM Articles, Specs, and White Papers

<http://www.microsoft.com/com/com.asp>

ZDNet's Webopedia Page of COM Links

[http://www.zdwebopedia.com/TERM/C/Component\\_Object\\_Model.html](http://www.zdwebopedia.com/TERM/C/Component_Object_Model.html)

COM vs. CORBA: A Decision Framework (white paper)

[http://www.quoininc.com/quoininc/COM\\_CORBA.html](http://www.quoininc.com/quoininc/COM_CORBA.html)

DCOM and CORBA Side by Side, Step by Step, and Layer by Layer

<http://www.cs.wustl.edu/~schmidt/submit/Paper.html>

Object News COM Pages

[http://www.objectnews.com/com\\_page.htm](http://www.objectnews.com/com_page.htm)

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## Feature Article

### Knowledge Bases On The Web

An Interview with Arnor Baldvinsson,  
Troy Sorzano, and Steve Parker

The growth of the Internet has meant massive increase in the amount of information available to programmers, much of it through newsgroups and web sites. Yet as Troy Sorzano points out in this week's [Rants and Raves column](#), the problem of finding the information you need remains acute. Three individuals, Troy Sorzano, Steve Parker and Arnor Baldvinsson, have each created and operate on-line databases of Clarion programming information. In this feature interview they explain how their databases came to be, how they work, and where they're headed next.

To see the knowledge bases in action, go to these links:

- [Steve Parker: CoolFAQs/Cheat Sheet](#)
- [Troy Sorzano: CWSuperKB](#)
- [Arnor Baldvinsson: IceTips](#)
- [Topspeed KB \(partnered with IceTips\)](#)

Where did the idea for the knowledge base start?

Arnor: I guess it all started as a brainstorming session between the left and right sides of my brain sometime before Easter last year (1998). At that time I had collected a few articles from the newsgroups but had nothing to really document them with. Kurt Pawlikowski emailed me in early April, and asked why I didn't write up a small program to keep online with articles etc. that could be searched. And stupid me, never thinking things are impossible to do, said "hmm... why not?" Kurt spoke to Steve Parker who graciously agreed to host the darn thing.

When going back and checking my emails from the time, it strikes me that those first three knowledge bases all came around at the same time. In late April, Troy contacted me. Then he was preparing for his EuroDevcon session about knowledge bases and was starting his own.

Troy: On January 9, 1998 at 2:43pm. This was the precise moment I posted a forum message to CompuServe outlining the reasons Topspeed needed a knowledge base. This post came about because of my frustrations with C4 Gold ODBC bugs and DAB's [Clarion Online](#) article titled "Zero Defects => Zero Productivity!" I felt that the entire Clarion community was full of developers wasting too much time on known bugs and workarounds because Topspeed did not have the proper resources available for the developers to help themselves solve the problems.

So the idea was planted in my head that Topspeed needed a Knowledge base. I did not really have the time to sit down and code one but the idea kept running through my head. My motivation to code the CWSuperKB was based on the reason most things get done—a deadline. I needed a knowledge base for my presentation at EuroDevCon '98, "Using CW to Create an On-line Knowledge Base for the Web."

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

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



Topspeed had released CWIC almost a year before EuroDevCon, but not many people were actually running applications on the Internet. There were lots of posts on the newsgroups asking where example CWIC applications could be found, so I knew that a working knowledge base would be a very useful example of what could be done with CWIC.

Like any good developer, I started coding the knowledge base about two weeks before EuroDevCon '98; nothing like a rock solid deadline to get me moving. Ironically, the weekend I started coding the KB I sent a message to Arnor asking if I could link my KB to some of his articles on IceTips. Arnor replied that he and Steve Parker had been working on a knowledge base and he would be going to EuroDevCon '98. He was interested in seeing my presentation, which was kind of funny since at that point I did not have a working KB. So it turns out that several knowledge bases were being built around the same time.

That weekend I created the CWSuperKB. The application logic and search engine only took about four hours; the rest of the weekend was spent working with the HTML and getting the look and feel just right.

Steve: The idea had actually been floating around for some time though Troy acted on it first. "Floating," in this case, means that lots of folks had been asking for one.

In my case, I got tired of all the, ah, grouching about "Topspeed should do this" and "Topspeed should do that." Several months earlier, I had offered to host a program written and maintained by someone else. This was a test; folks didn't realize just how tough this could be. There were no takers (no surprise). In the meantime, I had been accumulating tips and tricks from the newsgroups as I tried to learn my ABCs. The print outs were beginning to get cumbersome. So I wrote a little program to house them. Well, I found myself answering questions more than once, often by referring to my "stash." So, I took a few days, gussied my app up a bit (mostly hiding tabs), put the IC extensions on it and started referring folks to CoolFAQs.

I continue to collect tips and tricks that I think I will find helpful. But I've added another dimension: if I think something could become a repeated question, I "FAQ" it.

My concentration is on tips, tricks, how to items. I don't catalog workarounds or bugs. Since the bugs will probably go away, the workarounds will no longer be relevant. This lessens my maintenance while giving a longer life to articles.

At the same time, [CoolFAQs](#) (aka "Cheat Sheet") has provided me an opportunity to test and showcase IC's capabilities. There is a full Java version, almost completely "out of the box" IC, as well as several (sequential) Java-free versions. Interestingly, though both are readily available, two out of three hits are on the Java.

When did you go live?

Arnor: After a month of preparations and testing, IKB went online on May 10th last year.

Steve: All three of the KB's came on line in about a two week period. The order in which they were made public was: CoolFAQs, CWSuperPage and Icetips, though I think that CWSuperPage had actually existed for a few weeks before.

Troy: The first newsgroup posting I submitted about the CWSuperKB was on May 6, 1998. It was a stealth post as a reply to that dreaded Dynamic Pool Limit error. I posted the output of my knowledge base, listing two web resources for DPL errors; one was on <http://www.icetips.com> and the other was at <http://www.pointofsale.com>. Two days later, I made a formal announcement on the newsgroups about the CWSuperKB.

What were your concerns about using Java technology?

Arnor: From the first I decided that there would never be a Java browse in the IKB! There never has been. I have not once heard a complaint about it being slow either. At EuroDevcon I learned about the Java-free stuff that Tony Goldstein and Troy had used for their KB and Troy and I had some discussions about this KB thing in general. We had some initial problems with the program running on the server.

After some investigation I found out that it had started to misbehave when I got rid of the appframe as I thought it was useless because it was never called. But it turned out that something in the appframe was needed, so IKB has ever since used an appframe that calls the search window before the appframe opens and then goes directly out of the program. IKB used Java until December when I converted it to Java-free. I've done some bits and pieces here and there, but the basic file structures haven't changed until recently.

Troy: My major concern was not with Java but with the way TopSpeed used Java in their CWIC product. Java is very cool and a very powerful tool but, like any programming language, there is a time and place for it. I think that a vast majority of applications written for the web using CW do not require Java. Many people have said that CWIC is too slow and the technology is unusable on the Internet. Java was the main reason for causing these concerns about the CWIC product.

Even with the first version of the CWSuperKB that did not have any standard CWIC Java browses, my application still required downloads of the Java class files. I remembered that some of the early CWIC betas did not have any Java code in them, so I thought it would be possible to remove any references to the Java classes from CWIC. Looking at the HTML that was generated by CWIC, I could see some Java class references that were not being used by my code. Tony Goldstein—one of Information Packaging's developers—figured out how to remove all the references to Java in our CWIC templates. We call this template modification the Caffeine-Free CWIC technique. (To learn how to modify your CWIC templates, take a look at [Tony's Caffeine-Free CWIC article](#).)

Caffeine-Free CWIC was a big boost for CWIC developers, as before this every CWIC application required the Java class files. There appeared to be a resurgence of developer interest in CWIC once people saw it was possible to create Java-less apps with CWIC. The Caffeine-Free CWIC technique worked so well that even TopSpeed used it when they created the on-line registration form for DevCon '98.

Steve: I had some concerns; there had been a lot of complaints about

From: Troy Sorzano  
Newsgroups: comp.lang.clarion  
Subject: CWSuper Knowledge Base is in Beta  
Date: Fri, 08 May 1998 11:53:25 EDT  
Organization: Information Packaging Unlimited

Check it out  
[Http://www.cwsuperpage.com/kb](http://www.cwsuperpage.com/kb)

I created the CWSuperKB for the EuroDevCon TopSpeed conference next week.

The Clarion for Windows Super Knowledge Base.

The CWSuperKB is a Knowledge Base built to utilize all the Clarion resources on the web. By indexing articles, tip & tricks, FAQ's and files the Clarion community now has a way to SEARCH for knowledge. Every article or file in the CWSuperKB is a hypertext link to either a http document or ftp file. These documents and files are stored on servers all over the internet. If you have an article to add to the CWSuperKB just send email to [webmaster@cwsuperpage.com](mailto:webmaster@cwsuperpage.com) with the article title, keywords, short summary and web link. If your article is not currently published on the web send the article and it will be hosted for free on the CWSuperPage.

The CWSuperKB utilizes TopSpeed's CWIC and Sybase SQL Anywhere. It has been designed with the look and feel of a true web Knowledge Base not the standard "Windows" style applications that CWIC normally creates.

<http://www.cwsuperpage.com/kb>

Later,  
Troy

the slowness of Java applications. But Java got me up fast and provided an environment that would be very familiar to a Clarion developer. I built a Java-free version very soon thereafter. It's interesting, however, that whenever I check the broker's status, over two-thirds of the hits are on the full Java version. I've had only a few complaints about the speed.

When I built the [CWICWEB download site](#), which I consider a part of CoolFAQs in a way (CoolFAQs is for FAQs, CWICWEB for downloading and longer articles), I went totally Java-free. Totally. Viewed in the Clarion IDE, this app has only a button or two. Otherwise, it's a blank window. For its intended purpose, this is quite appropriate and effective (again, it is documented in [Clarion Online](#)).

What searching technology do you use?

Steve: Instring and eyeballs. Keyword searches are handled with INSTRING. But because Clarion developers are familiar with tabs and scrolling lists, I thought they would be immediately comfortable with an app using that technologies. This has proven to be true. In other words I depend on user intelligence (and haven't been disappointed yet).

Keyword searches can get a visitor to appropriate FAQs quite quickly. By putting the memo containing the FAQ on the browse window, visitors are also able to browse and glean information they might not have realized would be helpful. It seems to work.

Arnor: It was clear to me from the start that searching a KB with the built in functions wouldn't be very efficient. So I spent a lot of time doing a word parser that splits the articles into words that are stored in a keyword table. This parser was not good enough and has now been replaced with classes I wrote for this purpose. With these classes and some other tweaking the indexing process for 2400 articles has dropped from 45 minutes to nine and half minutes. This keyword table contains the word and a unique ID. Each article has also a unique ID and these two tables come together in a many-to-many relationship through a file that only stores these two IDs. In the Topspeed KB this is the second largest file, 4.5Mb compared to 4.6 for the articles file. It contains over 200,000 records that relate the keywords to the articles. When you type in a single word to search for, it does a GET() on the keyword table and if the word is not found it will list the keywords with the closest alphabetical and sound match, but if it exists it runs a range limited process against the relations table using the word ID as a range limit and stores the found article ID's in a global queue, which is then used to build the results page and the article page.

Troy: I used Sybase SQL Anywhere as the backend for the CWSuperKB. The searching is done with a keyword index. I originally wrote it to only support the OR operator but I woke up in my hotel room the morning of my EuroDevCon '98 presentation with an idea of how to add the AND operator. It turned out to be a simple two-line addition to my logic; isn't it amazing what can pop into your head when you're sleeping? I have not yet had the time to code any NOT logic into the search engine.

We are using the same basic search engine techniques in our current Intranet project for a very large insurance company. This Intranet application has a second-generation search engine that weighs the results based on where the search words are found. For example, keywords are weighted higher than words in the article title, which are weighted higher than words found in the article text. This has been working well for our clients.

How do you create the results page?

Steve: Straight HTML, no Java, no report using my favorite IC method, Target.Writeln. Much of the underlying technology has been written up and documented extensively in [Clarion Online](#).

Arnor: The results page with the article links is created completely dynamically. The window only contains the Close button and nothing else. I may be wrong, but as far as I know, IKB and Taufiq's IC program at [www.caramerica.com](http://www.caramerica.com) are the only two programs that use such completely dynamically created pages. Again, coincident strikes, as we

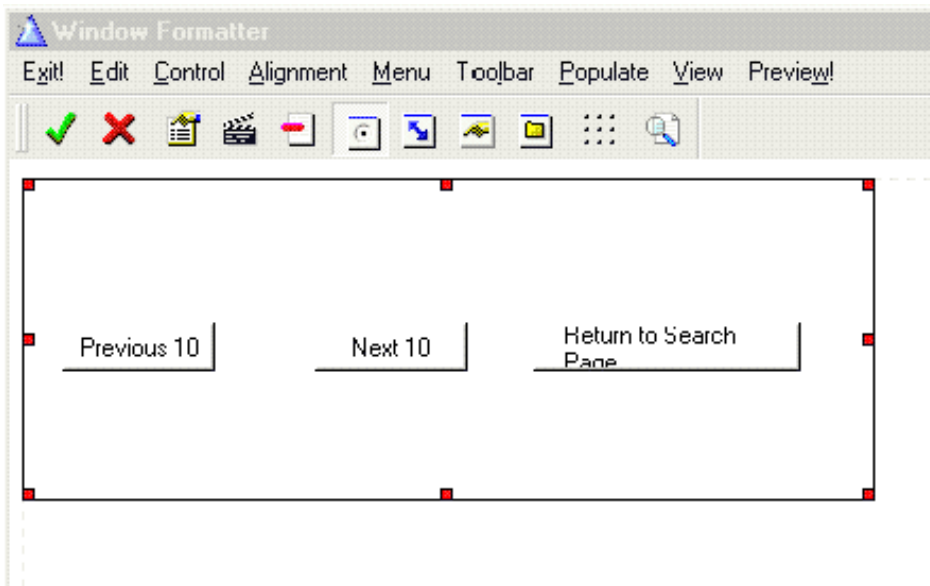
were both working on this over Christmas and shared some ideas and code via email.

The procedure that builds the article page GETS the article ID from the results page and then it's a single GET() on the article file. If you type in multiple words to search for or enclose the search string in double quotes, it uses a process and INSTRING() to search for the string in the articles. If you enclose words in double quotes you will notice that the search field is cleared when you return to it. This actually turned out to be a bug somewhere (I'm not sure where) that Jake Smith reported to us tonight that he had fixed. This is basically how the online program works. The online program is just a part of the package. Another program is used to do the file maintenance, importing articles, indexing, exporting html and whatever it can do.

A complete re-indexing of those 2400 articles that are in the Icetips/Topspeed knowledgebase takes around 9.5 minutes. It builds the keyword list which contains close to 20,000 words and the relations file which contains over 200,000 records. This program also connects articles to categories, allows us to edit the articles, checks the articles for too long lines etc. And I have just added a small module that attempts to analyse the keyword table to make it easier to find possible duplicates, misspelled words (you couldn't imagine how many different ways there are to spell Topspeed), and word frequency.

Troy: The result page is just like the Microsoft Knowledge Base result page; a plain HTML result page listing the articles with hypertext links and descriptions. The page in CW is a Window that contains the Next, Previous, and Return to Search Page buttons.

Figure 1. The CW Window with navigation buttons.



The result page HTML is created by building the HTML code on the fly and outputting it with the Target.WriteLine() statement.

Figure 2. The HTML code to create the results page.

```

Target.WriteLine('<<table cellpadding=0 cellspacing=3 border=0 width="95%">')

Loop X# = Glo:Page * Glo:ResultsToDisplay - (Glo:ResultsToDisplay-1) |
    to Glo:Page * Glo:ResultsToDisplay
    If X# > Glo:TotalHits then Break.
    Get(WebQ,X#)
    Target.WriteLine('<<tr width="100%"><td width=20 valign="TOP">'&|
        '<<font face="Verdana,Arial,Helvetica" size=2>')
    Target.WriteLine(Clip(WebQ:Count)&'<</td>')
    Target.WriteLine('<<td><<font face="Verdana, Arial, Helvetica" size=2>')
    Target.WriteLine('<<b><<a href=""&Clip(WebQ:Link)&"" target="_blank">'&|
        Clip(WebQ:Title)&'<</b><</a><</font><</td><</tr>')

    If Glo:TitlesOnly = 'N' then
        Target.WriteLine('<<tr><<td><</td><<td><<font face="Verdana,Arial,Helvetica" size=1>')
        Target.WriteLine('<<i>Excerpt from this page: <</i> '&Clip(WebQ:Summary)&'...<<cite>')
        Target.WriteLine('<<font face="Verdana,Arial,Helvetica" size=1>(updated '&|
            Format(WebQ:Date_Updated,@D10)&')<</font>')
        Target.WriteLine('<</cite><</font><</td><</tr>')
    End
End
target.WriteLine('<</table><<br>')

```

The following is what the users see in their browsers. Please note the above source only shows the code for the table containing the article results. The code for the image, query info, and the page footer is not shown.

Figure 3. The HTML page delivered to the user.

## Your search for *dynamic pool limit error* found the following 2 articles:

### 1. [ICETIPS: Solving the Dynamic Pool Limits by Geoff Robinson](#)

*Excerpt from this page:* Dynamic Pool Limits occur when compiling 16 bit applications. I have noticed fairly regular postings on Dynamic Pool Limit problems and have myself been doing battle with these up until quite recently but have now come up with a solution which works well for m...  
(Updated: April 29, 1998 Article ID#: 1000)

### 2. [Everything You Wanted To Know About Dynamic Pool Limits \(...but were afraid to a](#)

*Excerpt from this page:* Dynamic Pool Limits occur when compiling 16 bit applications. This is an extract from a session on a CW-Talk IRC chat session. Includes links to the DPLHELPER template that can help get rid of DPL errors from your application....  
(Updated: April 29, 1998 Article ID#: 1001)

[Return to Search Page](#)

Do you keep statistics on how many people use the KB?

Steve: I do not maintain formal stats on hits or utilization.

Troy: That is proprietary information. Do you want to buy my logs? Just kidding. I keep track of every query that is executed, IP address, date and time, and the number of returned articles. This allows me to scan the log file and see what people are looking for in my KB. I can also search my logs using SQL statements looking for queries that had returned no results. I can then do a little research on those topics and, if possible, add some articles to the database that would answer those user questions. There have been 16,379 queries executed since the KB was put online.

Arnor: Sometime in January I added a login database which logs all visits to the KB. I have a plan to temporarily set up a log of what is



being searched for, how many matches and more importantly what is being searched for that results in no matches at all. Analysing that data will help me figure out how people are using it and what searches fail, and I can then find out why and try to improve it accordingly.

Six weeks ago I got an email from Jim DeFabia saying that TS wanted to do something like this and since I had this up and running they didn't want to reinvent the wheel too much. They had basically two types of stuff that needed to be included; text files and bug reports. This meant some changes to the file structures to accommodate some checks and adding the bugs database. Importing the text files was fairly simple as I had that already in the stuff that imports the newsgroups messages and importing the text files was simpler as then I didn't need to parse the damn message headers.

Importing the bugs database was a question of a process and a few lines of code. Changing the generated html for the TS layout was a simple thing as right from the start I designed IKB so that the surrounding html is stored in two ASCII (HTML) files that are read into two queues when the program starts and these are simply looped through and added to the generated html before and after the IC generated html. Last Friday Jim and I were on ICQ while they got it up and running and that was pretty much it. IKB does currently not use the dataset that TS uses, because I was waiting for my ISDN to be installed (got it yesterday) as the upload jumped from around 3Mb to 10Mb. We have still to work out exactly how we synchronize the databases from now on, but I don't see that as a problem. (Editor's note: The Topspeed version of IKB is available from [Topspeed's home page](#).)

What about having users add their own entries to the database?

Steve: Many developers claimed (loudly) that they want this. Arnor, Troy and I corresponded privately about this and we all figured what the result would be. I went and volunteered to "prove the pudding." No one ever has written a FAQ (though Sean Wilson did send me three very good links to articles he'd written – this happened before I put up the entry form).

Arnor: Right from the start I decided that this was a one-way communication. That is users can't add anything or change anything; it's a read-only situation. From Steve's experience I'd say it was a right decision too as he added a form to his and I think there are one or two articles that have been added on-line. It was in high demand though for both, but I resisted.

Troy: Well, I coded all the web entry routines using CW and the [SnapSoft OCX](#) on the airplane to England. That was the biggest mistake of my flying career; I was so excited about making these web-based entry forms that I forgot to get any sleep, so I arrived in England at seven a.m. without having slept for over 20 hours. That was one rough day. (From now on when I fly to England I am going to catch some Zs!) At the conference I demonstrated my on-line entry forms, which allow web users to add articles that link to remote sites, for example, URLs on IceTips or the user's own web server. Or, they allow articles to be put directly into the SQL database. The web entry form also accepts HTML code so the user can nicely format the article with fonts, colors, graphics, and any other HTML tags.

Why is it not available on the web? Well, the answer is simply that I just have not had the time to install all the modifications I made before EuroDevCon '98. The CWSuperKB that is currently running is the original version that I posted on 5/6/98 before I flew to England; the only minor modification was made on 5/23/98, implementing the [Caffeine-Free CWIC](#) technique so the CWSuperKB would be running 100% Java free.

Has the response to the KB met your expectations?

Steve: Yes and no. It has helped me and others both learn and get our projects out the door. It has also helped develop and showcase the IC technology.

I've been able to answer questions on the newsgroups simply by

remembering that a subject has been "FAQ'd" (I always verify this first and supply a search string). I don't have to know what I'm talking about, only that the developer who posted the FAQ is reliable (which is easy to know in this particular community).

On the other hand, the number of times all of us have to refer to the Knowledge Bases indicates that they are not being as well utilized as they could be. Similarly, the lack of submissions indicates that the community is often willing to share other's solutions but not motivated to (or simply don't take the time to) share their own. I admit to being a bit disappointed here.

Arnor: IKB started as an IC experiment so I didn't have any particular expectations for it. As I said earlier, the IKB started more and less as a test project for Internet Connect. I'd been saving messages from the newsgroups for some time and didn't have anything to archive the messages with. The response to the knowledge bases has as far as I know, only been positive. That is certainly true for IKB at least, and that's enough for me.

None of this would have been possible if Steve P. and Bill H. hadn't taken IKB under their wings and hosted it. Special thanks to them! Alexey has also been very helpful (as always) providing ideas and code snippets to optimize the parser and searching as well as info on how the internals in some of the Clarion functions work and possibilities and algorithms to try to squeeze a few milliseconds here and there - it all adds up! Tony and Troy did something remarkable with IC! They ploughed the road for further IC development with their java free templates and classes. Thanks guys!

Troy: Its design has held up to the test of time. Many thanks to Microsoft and its thousands of employees who designed the original one on which the CWSuperKB was based. The CWSuperKB really pushed the limits of what people thought CWIC could do when it was first released; it did not use any Java browses and it was made to look and run just like a well-designed HTML web application. I have received many compliments from people inside and outside the Clarion community about how fast it is. We use it as an example of the Internet/Intranet applications that Information Packaging Unlimited can build for our clients.

What plans do you have for the future?

Steve: I'm looking forward to adding a tab for C6.

Arnor: Icetips Knowledge Base is now a joint project with TopSpeed and their knowledge base so plans for future versions of the program and how it evolves will have to be considered from more than just my viewpoint. Currently I'm working on getting the Icetips ftp site logged up to date into IKB, so that it can be updated much more frequently. I've also been working on advanced searching methods and improving indexing of the existing articles. Automating the indexing does have some problems as in many of the articles there is a lot of "noise" that shouldn't really be indexed and I'm experimenting with some methods to eliminating that noise.

Troy: You had to ask didn't you. I had no plans for upgrading before you asked me these questions. But my subconscious started thinking and after a few days these ideas popped out. First I will need to get the web article submission running on our server and also upgrade the search engine to our second generation version with weighted search results. Once that is done I can start working on some of these new features, including a HTTP Internet agent that will automatically index user submitted articles that are hosted on remote servers. The user can then verify the keywords chosen instead of my having to enter them by hand.

The second idea I had was another HTTP Internet agent that will verify all the links in the CWSuperKB database once a week. At the same time it will check the last modified date of the page, thus allowing the CWSuperKB to list the most recently updated sites as part of its search options.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).





# Clarion magazine

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

## The Clarion Challenge

### The Clarion Code Challenge: A Compiler's Eye View

#### Part One

By David Bayliss

(Editor's note: Some of the source code on this page exceeds the page boundaries when printed from a browser. You may wish to [download the April 19 PDF](#) which takes care of this problem.)

Recently Clarion magazine ran a competition to see who could produce the fastest and tightest code for a given problem. The results are given elsewhere but David has asked me to write an article dealing not with what the developer was doing, but what the compiler was doing underneath.

The object is not to score points from one developer to another but simply to use the code that has been produced to illustrate some of the internals of the way the compiler works. These are not issues you would want to think about when you code but, if they are in the back of your mind, they may just help you keep an edge on your coding.

In order to make the analysis concrete I shall actually include snippets of disassembled code from the functions. Don't worry if you're not an assembler coder. The text has all the information required, and this may even ease you in to using some of the debugger disassembly functions. The 32 bit debugger is brilliant in mixed code/assembler mode. I have also given a byte count for each function. The byte count is often an indication of just how easy the compiler found the code to chew. Generally I look for around 20 bytes per line as a fair indication that the code is doing what you expect.

As well as analysing what the compiler has produced I shall make a few comments about how I might have altered the code provided. These comments may not actually help (they are not tested), but they are the comments I would have made to one of my team had they submitted the code as part of our code base.

Jeff Slarve – 284 bytes

```
JeffSlarve Procedure(Long StartTime,Long EndTime,
    Short Increment,Byte Round=False)
Result Real
Code
Result = ((EndTime - StartTime + |
    CHOOSE(StartTime > EndTime, 8640000, 1)) |
    / Increment / 60) * .01
Return CHOOSE(~Round,Result+CHOOSE|
    (~(Result-Int(Result)),0,1),Round(Result,1))
```

**NextAge**  
Consulting  
**Imaging**  
**Templates**

Add Scanning and  
Document Storage  
to your app  
in 10 minutes

**Only \$149**

**BKO**  
Enterprises, Inc.

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

**etc**  
**2000**

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

Jeff is one of the many that went for minimum line count. The Clarion CHOOSE function is central to this philosophy and the fragment below shows why.

```
!39 C6!           cmp esi, eax
!7E 07!           jle 7
!BB 00 D6 83 00!  mov ebx, 83D600H
!EB 05!           jmp 5
!BB 01 00 00 00!  mov ebx, 1
```

This is the code for the inner CHOOSE of the first line. The compiler has the two incoming LONGS in registers, compares them and stores 8640000 or 1 in ebx dependent on the equality.

Adding EndTime and then subtracting StartTime becomes a short and sweet operation.

```
!29 F0!           sub eax, esi ! eax = EndTime-StartTime
!01 D8!           add eax, ebx ! eax += CHOOSE result (in ebx)
```

Having got the body of the expression into eax the compiler now has to divide by Increment. Here is the first issue. In Clarion division actually works. A division (even of an integer by an integer) is defined to produce a decimal result. So now the compiler uses the decimal stack. The first point to note is that the compiler didn't compute the result in eax by accident. It was to make it easy to push the value onto the decimal stack.

```
!E8 00 00 00 00!  call Cla$DpushLong ! Push result
!0F BF C1!         movsx eax, cx      ! Push increment onto stack
!E8 00 00 00 00!  call Cla$DPushLong
!E8 00 00 00 00!  call Cla$DecDivide ! Perform division, result on stack
```

To get increment onto the stack it has to work a little harder. First increment ( a SHORT in cx ) is moved into eax with sign extension, then the push happens, then the divide. Next the compiler has to divide by 60. Here is a nice feature of the decimal stack: the result of the division was left on the stack so to divide by 60 it's only necessary to push one of the operands (not both).

```
!B8 3C 00 00 00!  mov eax, 3CH      ! 60 onto stack
!E8 00 00 00 00!  call Cla$DPushLong
!E8 00 00 00 00!  call Cla$DecDivide
```

Next comes the multiplication by .01. Another little tweak: the compiler itself does not have a decimal library built in so cannot perform decimal arithmetic. Thus it stores decimals internally as strings and hands them on to the decimal library as such. Again the result of the division is still on the stack so the constant is pushed, then the multiply routine is called.

```
!B8 00 00 00 00!  mov eax, $7
!B3 02!           mov bl, 2
!66 B9 02 00!     mov cx, 2
!E8 00 00 00 00!  call Cla$DpushConstant ! .01 on stack
!E8 00 00 00 00!  call Cla$DecMul      ! x top two items
```

Finally on the first line is the assignment itself. This is made more complex (at least in the assembler) by the fact that the result variable is a REAL. To store one of those the floating point chip is used. `DpopReal` moves a result from the decimal stack to register 0 of the FPU. This is then stored as required.

```
!E8 00 00 00 00!          call Cla$DpopReal
!DD 5C 24 30!            fstp qword [esp][30H],st0
```

In the second line a `CHOOSE` is again evaluated (the outer one). Simply checking a byte for true is very simple. Note that the compiler doesn't actually evaluate `~Round`; it simply reverses the jump condition. The other key here is that `CHOOSE` is guaranteed to perform a short-circuit evaluation. In other words only one of the last two parameters of `CHOOSE` will be evaluated. This is vital! Without it all solutions using `CHOOSE` would have been much slower.

```
!80 FA 00!              cmp dl,0
!75 4C!                 jne 4CH
```

Going down the non-rounding arm the compiler first gets the evaluation of `~(Result-INT(Result))`. This can be done almost entirely on the FPU although the Clarion `INT` semantics have been encoded in a run-time library function. Note too that it's necessary to use the Windows calling convention which means floating point numbers are passed on the stack. (But they are returned in `FP0`! Don't ask.)

```
!83 EC 08!              sub esp,8
!DD 44 24 40!            fld st0,qword [esp][40H]
!DD 1C 24!               fstp qword [esp][0],st0
!E8 00 00 00 00!          call Cla$INT                ! INT(Result)
!DD 5C 24 28!            fstp qword [esp][28H],st0
!DD 44 24 38!            fld st0,qword [esp][38H]
!DD 44 24 28!            fld st0,qword [esp][28H]
!DE E9!                 fsubp st1,st0              !Result-INT(Result)
!DD 05 00 00 00 00!       fld st0,qword [dword $8]    ! Load up 0
!DE D9!                 fcompp st0,st1            ! Compare against zero
```

Having computed the result into the FPU the compiler now has to transfer the results into the normal registers and then execute the `CHOOSE`. The `CHOOSE` will return 0 or 1 (both of which are longs).

```
!DF E0!                 fstsw ax                    ! FPU status goes into eax
!F6 C4 40!              test ah,40H
!74 07!                 je 7
!B8 00 00 00 00!          mov eax,0
!EB 05!                 jmp 5
!B8 01 00 00 00!          mov eax,1                    ! EAX now holds 0 or 1 from CHOOSE
```

Having the result of the `CHOOSE` in `eax` it now has to be cast back into a `REAL` to perform the addition:

```
!DD 44 24 38!            fld st0,qword [esp][38H]    ! Result into fpu
!89 44 24 24!            mov [esp][24H],eax
!DB 44 24 24!            fld st0,dword [esp][24H]    ! EAX into fpu
!DE C1!                 faddp st1,st0              ! Addition performed
!DD 5C 24 1C!            fstp qword [esp][1CH],st0    ! Answer now ready for
returning.
```

The computation of the other arm is simpler as the result of a ROUND is deemed to be dependant upon the second (not first) parameter. Thus the decimal stack is used.

```
!83 EC 08!           sub esp,8
!DD 44 24 40!       fld st0,qword [esp][40H]
!DD 1C 24!          fstp qword [esp][0],st0
!E8 00 00 00 00!   call Cla$DpushReal  ! Result onto decimal stack
!B8 01 00 00 00!   mov eax,1
!E8 00 00 00 00!   call Cla$DpushLong  ! 1 onto decimal stack
!E8 00 00 00 00!   call Cla$Dround     ! Rounding performed
!E8 00 00 00 00!   call Cla$DpopReal   ! Result back in st0
```

The last line may seem a bit odd. Why bring the result back as a REAL? A CHOOSE function has to return the same result type whichever arm is executed, so the compiler chooses the lowest common denominator as the final result type. In the case of REAL and DECIMAL this is deemed to be REAL.

Having computed the result as a REAL it actually has to be returned as a SHORT. This can be a real performance hit as the FPU cannot be guaranteed to have the correct truncation mode set. Don't ask what the lines below do. I knew it five years ago but have long since forgotten. I do know we need them.

```
!DD 44 24 1C!       fld st0,qword [esp][1CH]
!D9 7C 24 10!       fstcw [esp][10H]
!D9 7C 24 0E!       fstcw [esp][0EH]
!66 81 4C 24 0E 3F 0C! or word [esp][0EH],0C3FH
!D9 6C 24 0E!       fldcw [esp][0EH]
!DF 5C 24 12!       fistp word [esp][12H],st0
!DB E2!            fclex
!D9 6C 24 10!       fldcw [esp][10H]
!66 8B 44 24 12!    mov ax,[esp][12H]
```

## Summary

The compiler was quite happy with this code, and the CHOOSE made the execution path short and quick. The only thing that made the assembler ugly was the use of the real. Generally I would use a decimal rather than a real when mixing with longs. A small savings could also be achieved by using ,AUTO on the variable.

David Bayliss – 147 bytes

```
DavidBayliss PROCEDURE(s_time,e_time,incr,rnd)
dur long
CODE
dur = e_time-s_time
IF dur < 0 THEN
  dur += 2 * 12 * 60 * 60 * 100
.
IF rnd
  RETURN ROUND(dur / incr / ( 60 * 100 ),1)
ELSE
  dur /= incr * 60 * 100
  RETURN dur + 1
END
```

My main aim here was to stay at long precision for as much of the time as I could. Hence my only intermediate is a LONG. (Yuk, sloppy. This should have had ,AUTO on it). I compute the duration in the first line; this can be done easily as the incoming parameters are in registers.

```
!29 C3!           sub ebx,eax           ! ebx is now dur
!89 5C 24 04!     mov [esp][4],ebx     ! stored in memory
```

The comparison of a LONG with zero is very cheap as the compiler doesn't even bother to load the value from memory. The compiler is also very good at folding integral expressions so I didn't bother to fold the multiplication by hand. I used += as this avoids using the value of dur and hence avoids a load into a register. The whole of the first IF construct is below.

```
!83 7C 24 04 00!   cmp dword [esp][4],0
!7D 08!           jge 8
!81 44 24 04 00 D6 83 00! add dword [esp][4],83D600H
```

Then comes the actual computation of the result. The rounding arm comes first. Here I missed a trick which cost me the poor rounding result.

```
!8B 44 24 04!     mov eax,[esp][4]
!E8 00 00 00 00!   call Cla$DpushLong ! dur onto decimal stack
!89 D8!           mov eax,ebx
!E8 00 00 00 00!   call Cla$DpushLong ! increment onto decimal stack
!E8 00 00 00 00!   call Cla$DecDivide ! Divide and leave on stack
!B8 70 17 00 00!   mov eax,1770H
!E8 00 00 00 00!   call Cla$DpushLong ! 60 * 100 onto stack
!E8 00 00 00 00!   call Cla$DecDivide ! Divide and leave on stack
!B8 01 00 00 00!   mov eax,1
!E8 00 00 00 00!   call Cla$DpushLong ! One onto stack
!E8 00 00 00 00!   call Cla$Dround    ! Round it
!E8 00 00 00 00!   call Cla$DpopLong  ! Return as short
```

The compiler has coped with the expression just fine, but the trick I missed is that I could have avoided the second division by using a multiplication instead.

```
RETURN ROUND(dur / (incr * ( 60 * 100 )),1)
```

The reason this matters is the dur/incr can often leave a result with many decimal places. The division by 6000 (which the decimal library does to 31 decimal places) is thus very costly.

The non-rounding arm actually missing a trick too.

```
!8B 44 24 04!     mov eax,[esp][4]
!E8 00 00 00 00!   call Cla$DpushLong ! Dur onto decimal stack
!6B C3 3C!        imul eax,ebx,3CH   ! Incr (in ebx) times 60, into eax
!6B C0 64!        imul eax,eax,64H   ! Eax times 100 into eax
!E8 00 00 00 00!   call Cla$DpushLong ! Result onto decimal stack
!E8 00 00 00 00!   call Cla$DecDivide ! Division performed
!E8 00 00 00 00!   call Cla$DpopLong  ! Result into eax
!89 44 24 04!     mov [esp][4],eax   ! Result into dur
```

So how come the silly compiler didn't fold my constants? Well, it was actually being ultra-cautious. In a fixed-width number system (such as long or decimal) multiplication is actually not quite associative. If you rewrite the line as

```
dur /= incr * (60 * 100)
```

the compiler folds as expected. Moral: If you want to make sure constants are folded in a mixed expression then use some parenthesis.

Adding on one and return is then easy:

```
!66 8B 44 24 04!           mov ax,[esp][4]           ! Long to short cast is free
!66 40!                   inc ax                   ! Adding one to a long is easy
```

## Summary

A painfully sloppy piece of coding. I missed three tricks all of which cost time, one of which cost compactness. However, I actually find this quite encouraging. You see, when I sent this to David I said that I hadn't checked for speed or compactness; I was just trying to code the function "cleanly." The result was a function that performed reasonably well and can be cleaned up into a very nice performer.

Nik Johnson A – 160 bytes

```
NikJohnsonA Procedure(Long T1,Long T2,Short B,Byte R=False)
Result DECIMAL(4)
CODE
Result = .5-R/2+(T2-T1-1+CHOOSE(T2-T1<1,8640000,0))/(B*6000)
return(Result)
```

Here again the decimal stack is in full swing. First R is converted to a LONG and divided by two (which is a decimal operation).

```
!0F B6 C2!           movzx eax,dl           ! R (in dl) goes into EAX
!E8 00 00 00 00!     call Cla$DpushLong    ! And onto the decimal stack
!B8 02 00 00 00!     mov eax,2
!E8 00 00 00 00!     call Cla$DpushLong    ! 2 onto decimal stack
!E8 00 00 00 00!     call Cla$DecDivide    ! Division performed,
!                                     ! result on stack
```

Then the 0.5 is pushed onto the stack

```
!B8 00 00 00 00!     mov eax,$5           ! 0.5 stored as string
!B3 01!             mov bl,1
!66 B9 01 00!       mov cx,1
!E8 00 00 00 00!     call Cla$DpushConstant ! 0.5 on stack
```

Then the subtraction. This shows a useful tweak in the stack system. Usually a stack subtraction would take the top of stack from the one below, the opposite of what we want. We could issue some "stack rolling" calls, however we have avoided this by producing some "reversed" functions (that actually want their parameters inside out).

```
!E8 00 00 00 00!     call Cla$DecSubR      ! Take 2nd on stack from top,
!                                     ! result on stack.
```

The compiler leaves this result on the stack, and it will come back to it much later.

Now the compiler moves onto to the inner CHOOSE (very similar to Jeff Slarve's code).

```
!89 E8!           mov eax,ebp
!29 F0!           sub eax,esi      ! EAX is T2-T1
!83 F8 01!        cmp eax,1        ! Compare to 1
!7D 07!           jge 7
!BB 00 D6 83 00!  mov ebx,83D600H
!EB 05!           jmp 5
!BB 00 00 00 00!  mov ebx,0        ! EBX is result of CHOOSE
```

The  $T2-T1-1$  is all long arithmetic which you would expect the compiler to chew easily. In fact it is ultra-clever. Note that in computing the CHOOSE  $T2-T1$  is computed into `eax`, it so happens that  $T2-T1$  is the first part of the expression it needs to compute so it doesn't have too! It can get straight on with the subtracting of one and then adding the CHOOSE result.

```
!48!             dec eax          ! EAX = T2-T1-1
!01 D8!          add eax,ebx      ! Add on CHOOSE result.
```

The compiler has again chosen to compute the result into `eax`. You'll see why in the next stage which is the division of  $B * 6000$ .

```
!E8 00 00 00 00!  call Cla$DpushLong ! Result onto decimal stack
!0F BF C7!        movsx eax,di      ! B into eax
!69 C0 70 17 00 00!  imul eax,eax,1770H ! Multiply by 6000
!E8 00 00 00 00!  call Cla$DpushLong ! Onto decimal stack
!E8 00 00 00 00!  call Cla$DecDivide ! Division
```

Finally this result has to be added to the result of the subtraction performed previously. Remember that had been left on the stack so the compiler simply does:

```
!E8 00 00 00 00!  call Cla$DecAdd
```

This result is then placed into the local variable:

```
!8D 44 24 15!    lea eax,[esp][15H]
!B3 03!          mov bl,3
!B1 00!          mov cl,0
!E8 00 00 00 00!  call Cla$DpopDec
```

The local variable is then converted to a `SHORT` (via the stack) and returned.

```
!B3 03!          mov bl,3
!B1 00!          mov cl,0
!E8 00 00 00 00!  call Cla$DPushDec
!E8 00 00 00 00!  call Cla$DpopLong
```



## Summary

This function is very similar to Jeff's except the computation stays with DECIMALS rather than diving into REALS. The benefit is seen in terms of code size and speed. I think there are two ways to improve it. Firstly changing  $0.5-R/2$  to  $(1-R)/2$  would save the "constant push" and would allow the subtraction to happen at LONG width. Secondly I would try ditching the DECIMAL variable and simply returning a ROUNDED result. The DpopDec contains an implicit ROUND so I would guess this will not slow things down and would save some code.

Nik Johnson B – 153 bytes!

```
NikJohnsonB Procedure(Long T1,Long T2,Short B,Byte R=False)
Result DECIMAL(4)
CODE
Result = (((T2-T1)+CHOOSE((T2-T1)<0,8639999,-1)) |
/B/3000)+1-R)/2
return(Result)
```

The innermost expression (up to the first /) is a joy to behold. Again the CHOOSE is neat and the compiler repeats the trick of spotting the T2-T1 is used twice and so computes it once.

```
!29 F0!          sub eax,esi          ! EAX = T2-T1
!83 F8 00!       cmp eax,0
!7D 07!          jge 7
!BB FF D5 83 00! mov ebx,83D5FFH
!EB 05!          jmp 5
!BB FF FF FF FF! mov ebx,-1          ! EBX is result of CHOOSE
!01 D8!          add eax,ebx          ! Result of addition
```

Again the result is in eax which makes the following two divisions very neat.

```
!E8 00 00 00 00! call Cla$DpushLong  !Onto decimal stack
!0F BF C1!       movsx eax,cx        ! B onto decimal stack
!E8 00 00 00 00! call Cla$DPushLong
!E8 00 00 00 00! call Cla$DecDivide ! Division done
!B8 B8 0B 00 00! mov eax,0BB8H
!E8 00 00 00 00! call Cla$DpushLong ! 3000 onto stack
!E8 00 00 00 00! call Cla$DecDivide ! Division again
```

Now, here the compiler has a decimal result to which it wants to add 1 (again addition & subtraction are not quite associative so the compiler works left to right). Because it cannot cast from DECIMAL to LONG without losing precision it has to perform the addition (and then subtraction) upon the decimal stack.

```
!B8 01 00 00 00! mov eax,1
!E8 00 00 00 00! call Cla$DpushLong ! 1 onto stack
!E8 00 00 00 00! call Cla$DecAdd    ! Add
!0F B6 C2!       movzx eax,d1       ! R into EAX
!E8 00 00 00 00! call Cla$DpushLong ! and onto stack
!E8 00 00 00 00! call Cla$DecSub    ! Subtract
```

On the plus side the decimal computation leaves the result on the stack, ripe for the division by 2, again on the decimal stack.

```
!B8 02 00 00 00!      mov  eax,2
!E8 00 00 00 00!      call Cla$DPushLong
!E8 00 00 00 00!      call Cla$DecDivide
```

The result is then used as in Nik's A version.

## Summary

The assembler actually looks quite nice, certainly very compact. So why was it so slow? I suspect the culprit is the three divisions. The problem of doing repeated divisions within one expression is that the intermediate result gets very long which slows the decimal library down. This could be reduced by combining  $/B/3000$  into  $/(B*3000)$ .

The other little tweak would be to move the +1-R to the beginning of the bracketed expression so that it is done at long width.

Brian Staff – 211 bytes

```
BrianStaff Procedure(Long T1,Long T2,Short Block,Byte Rounding=False)
X          LONG
Answer    long
code
X = CHOOSE(T2 > T1,(T2-1) - (T1-1),|
(T2-1) + (24*60*60*100) - (T1-1))
EXECUTE(ROUNDING+1)
  Answer = ( X / (Block*60*100) ) + |
  CHOOSE(X % (Block*60*100) = 0,0,1)
  Answer = ROUND( X / (Block*60*100), 1)
END
return(Answer)
```

The compiler has really bust a gut getting the code for Brian's CHOOSE good. The jumping code is clean (as usual) but it also spots that T1-1 is used down both arms and thus pre-computed it. Here is the code for the whole line:

```
!39 C3!      cmp  ebx,eax      ! Test T2>T1
!9C!        pushfd
!48!        dec  eax      ! Pre-Compute T1-1
!9D!        popfd
!7E 05!     jle  5      ! Then pick an arm
!4B!        dec  ebx      ! EBX = T2-1
!29 C3!     sub  ebx,eax     ! EBX = (T2-1) - (T1-1)
!EB 08!     jmp  8
!81 C3 FF D5 83 00! add ebx,83D5FFH ! EBX = T2-1+24*60*60*100
!29 C3!     sub  ebx,eax     ! EBX is required result
!89 5C 24 0C! mov  [esp][0CH],ebx ! Store into X
```

Brian has also persuaded the compiler to pre-compute the  $Block*60*100$  outside of the execute statement.

```

!0F B6 D2!           movzx edx,dl           ! Move ROUNDING into edx
!42!                 inc edx               ! Add 1
!83 FA 01!           cmp edx,1             ! Compare against 1 (first line)
!9C!                 pushfd
!0F BF C1!           movsx eax,cx          ! Move Block into EAX
!6B C0 3C!           imul eax,eax,3CH     ! * 60
!6B D8 64!           imul ebx,eax,64H     ! EBX now holds Block*60*100
!9D!                 popfd
!75 4E!              jne 4EH              ! Now split into the two arms of the
execute.

```

The first part of the non-rounding case (the division) is handled in the way you should now expect

```

!8B 44 24 0C!       mov eax,[esp][0CH]! X into EAX
!E8 00 00 00 00!    call Cla$DPushLong
!89 D8!             mov eax,ebx          ! Compiler has pre-computed
Block*60*100
!E8 00 00 00 00!    call Cla$DPushLong
!E8 00 00 00 00!    call Cla$DecDivide ! Result as required

```

A modulus operation between integers is handled by the compiler inline (although the code is a little hard to understand unless you sit down with pencil and paper!).

```

!8B 44 24 0C!       mov eax,[esp][0CH] ! X into EAX
!99!                 cdq                  ! Make X an INT64
!F7 FB!             idiv ebx             ! Modules with Block*60*100
(Pre-computed)
!89 D0!             mov eax,edx          ! You now need to fiddle to deal with
negatives
!31 D8!             xor eax,ebx
!7D 06!             jge 6
!85 D2!             test edx,edx
!74 02!             je 2
!01 DA!             add edx,ebx          ! Get the result into edx by magic
(Memory failure ...)

```

Having got the result magically into edx the CHOOSE is its usual snappy self:

```

!83 FA 00!           cmp edx,0
!75 07!             jne 7
!B8 00 00 00 00!    mov eax,0
!EB 05!             jmp 5
!B8 01 00 00 00!    mov eax,1

```

This leaves the addition. The result being added to is presently on the decimal stack so decimal addition is used, and the result is then stored in Answer.

```

!E8 00 00 00 00!    call Cla$DpushLong ! CHOOSE result in EAX, now pushed.
!E8 00 00 00 00!    call Cla$DecAdd    ! Add to division result already on
stack
!E8 00 00 00 00!    call Cla$DpopLong  ! Pop result into eax
!89 44 24 08!       mov [esp][8],eax   ! And into answer

```

The other arm is fairly standard decimal stack stuff.

```

!8B 44 24 0C!      mov eax,[esp][0CH]  !X into eax
!E8 00 00 00 00!   call Cla$DpushLong! X onto stack
!89 D8!           mov eax,ebx        ! Pre-computed Block*60*100
!E8 00 00 00 00!   call Cla$DPushLong
!E8 00 00 00 00!   call Cla$DecDivide ! Division result on stack
!B8 01 00 00 00!   mov eax,1         ! 1 onto stack
!E8 00 00 00 00!   call Cla$DpushLong
!E8 00 00 00 00!   call Cla$Dround   ! Perform the round
!E8 00 00 00 00!   call Cla$DpopLong ! Result into eax
!89 44 24 08!     mov [esp][8],eax  ! And into answer

```

## Summary

Brian has kept the computations at a suitable precision, avoided excessive divisions and somehow persuaded the compiler to perform some extremely useful pre-computations. A worthy winner.

I'm not sure I should comment given he has the best figures but I think he could get a better result yet by putting the  $60*100$  into parenthesis (same as David Bayliss' code). He can also put `,AUTO` on his variables. The other thing I would try is returning the result directly rather than going through `Answer`. (Editor's note: Some entries were provided as routines and the extra step in `RETURNING` values may be mine.)

---

[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.

Part Two of this analysis (coming soon) provides some surprising information about how the compiler translates the code we write. Stay tuned.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## Search Clarion Magazine

Clarion Magazine is fully searchable and the search engine will return links to public access and subscriber-only articles. The search database is updated shortly after a new issue has been posted.

Match:

Format:

Sort By:

Matches Per Page:

Search For:



# Clarion magazine

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

## Rants & Raves

### How To Waste Time In The Newsgroups

By Troy Sorzano

Newsgroups were and are TopSpeed's "preferred" product support method. Yes, newsgroups do work and the Clarion community has been limping along using them for the last several years. But newsgroups are not efficient.

Think of a newsgroup as a one-to-many relationship: a person can post a quick question that only took two minutes to type up, and many people can and will read that post on the newsgroup. The problem is that no one knows how many people actually read that question or how much time they spent reading.

Just for an example assume that 100 people read that question, taking 20 seconds each. The question that only took two minutes to create has now utilized 33 minutes of the Clarion community's time. Taking it one step farther, what if one or more people respond to that question? How much time is then spent to resolve that two-minute question? The answer is "I don't know," but I do know I have sometimes spent anywhere from five minutes to two hours on some replies.

A newsgroup's inefficiency is that it allows a single resource to utilize multiple resources. This becomes an even bigger issue when a similar question is asked over and over again. How much time is wasted regurgitating the same answers to common questions? If you want to see an example, do a search on Dynamic Pool Limits using [DejaNews' power search](#) for the comp.lang.clarion newsgroup. People have been posting the same question over and over and over for years.

As far as I can tell, the first solution to Dynamic Pool Limit errors was posted on April 28, 1996, so there has been a solution on the newsgroup for the last three years. Why are people still asking the same Dynamic Pool Limit question in 1999? A search will show that the most recent question posted asking about pool limits was on January 4, 1999. There have been about 500 messages on comp.lang.clarion about this one error since the first solution was posted. The amount of time that has been spent on the Dynamic Pool Limit problem is mind-boggling. And the sad part is that there has been a posted solution for almost as long as the question has been asked.

Now some may argue that newsgroups are efficient because they can be searched. That may true for public newsgroups like comp.lang.clarion, but these days most of the Clarion discussion takes place on TopSpeed's own news server (tsnews.clarion.com), which is not picked up by usenet search agents like DejaNews. To search the TopSpeed newsgroups you would need to actually download all twenty thousand plus messages first. Even then, your searches may still run into trouble because you are searching through raw text. For example, try using DejaNews to search the comp.lang.clarion newsgroup for articles on "SQL Anywhere." You'll find a large number of messages that don't contain any information at all on SQL Anywhere, but instead mention it only in the poster's signature block.

etc  
2000

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

**BKO**  
Enterprises, Inc.

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

I hope you can understand that no matter how cool or neat we think newsgroups are, they are not a very efficient way to share the knowledge of the Clarion community. From a resource perspective, knowledge on a newsgroup is very expensive. Just imagine all the cool cutting-edge stuff someone like Arnor Baldvinsson could be coding if he was not answering dozens of newsgroup questions every day.

---

Troy Sorzano is a member of Team Topspeed Internet and a partner at [Information Packaging Unlimited](#) where he creates client/server and Internet applications in Clarion. He can be reached via e-mail at [troyweb@infopackaging.com](mailto:troyweb@infopackaging.com).

If you have a rant or a rave you'd like to get off your chest, send it to [editor@clarionmag.com](mailto:editor@clarionmag.com).

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).





# Clarion magazine

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

## Clarion Magazine PDFs

As of April 12, 1999 current articles will be available in PDF format as well as HTML format. Articles published prior to April 12, 1999 will be converted at a later date.

### To Download These Files

If your browser is configured to display PDF files and you want to download them instead, right-click on the appropriate link (see below) and choose the Download, Save Target As, Save Links As, or perhaps just the Save As option from the context menu. Most browsers support this feature.

### Restrictions On Use

As noted in the [subscription agreement](#), these PDFs are for your own use only as a valid subscriber. You may not distribute them.

### Which Acrobat Reader Do I Need?

These documents were created with Adobe Acrobat 4.0. There have been some reports of problems with 3.x readers, in which case you may need to upgrade to the 4.0 reader which you can [download for free from Adobe](#). At just over 5 megs this is a fairly hefty download. The 4.0 reader is also available at [Clarion Magazine](#) and at the [CWICWeb download page](#).

Because of some reported problems with 3.x readers PDFs are currently created without the background image you normally see on the web pages.

### Where Is This Week's PDF?

The current week's PDF is created shortly after the HTML documents are published on the web site, so if you've come here from the current issue and don't see the PDF yet, check back in an hour or two and be sure to reload/refresh the web page. If you still don't see the PDF after refreshing the page please send an email to [webmaster@clarionmag.com](mailto:webmaster@clarionmag.com).

### This Month's PDFs

April 5, 1999

<http://www.clarionmag.com/v1n3/sub/cmaga-april5-99.pdf>

April 12, 1999

<http://www.clarionmag.com/v1n3/sub/cmaga-april12-99.pdf>

April 19, 1999

<http://www.clarionmag.com/v1n3/sub/cmaga-april19-99.pdf>

April 26, 1999

<http://www.clarionmag.com/v1n3/sub/cmaga-april26-99.pdf>

## Archive PDFs

February 1999

<http://www.clarionmag.com/v1n1/pub/cmag-february-99.pdf>

March 1999

<http://www.clarionmag.com/v1n2/sub/cmag-march-99.pdf>

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine



by Andrew Guidroz II

"Glad to hear you're starting a new magazine, Dave. If there is anything I can do to help, just say the word."

"Well, since you're the official Clarion Cajun chef, how about writing up a DevCon Gumbo recipe?"

And just like that, my mouth got me in trouble again.

Here we are, two months after the launch of Clarion Magazine, and still no recipe. Is it writers' block? Lack of a good subject? I have nothing to say? Me???! King of thread drift? Nope. Dave with one simple request asked for the hardest thing a cook can do.

You see, it isn't so simple, writing a recipe – especially one of mine. By now, everyone around the Clarion internet community knows I'm a Cajun and I cook. I'm no chef and I'm certainly more gourmand (a pig) than gourmet. I cook in my own style that has been influenced by the cooking of my parents and their parents. But how do I convey what I'm doing in words? How do you teach a non-Cajun what color a roux should be? When I say, "Put three onions in" does the person on the other end know what an average sized onion looks like or whether it should be yellow, white, or purple? Does the person reading my recipe know he/she should peel and chop them?

Each cook out there has different levels of experience and expertise. And certainly the regional influence of my cooking adds one more level of complexity to all of this. I really can't write a generic recipe that works for everyone. I write one style for my Cajun friends that cook. I write another for my Cajun friends who don't. And I write in a variety of styles for people who may or may not cook and may live all over the world. But in every recipe I write there is a common goal: each should give clear directions to produce a good meal. Each recipe might have a different route but they all end up in the same place.

This problem isn't so different from the problem we all have when we write software. We tend to write our software with a certain end user or two in mind. But many different people of various skill levels and backgrounds will use our software. They are all trying to get to the same end product. Software is just a tool, a recipe, to take our users from the point of having nothing but inputs (ingredients) and moving toward a finished product: the gumbo of their completed data. Software is really just a dynamic, intelligent recipe and the data stored in your files is the meal, steaming hot and ready to eat.

In theory, this is all great. But certainly no one is going to write separate applications to distribute to different users. How can one application be

etc  
2000

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

**BKO**  
Enterprises, Inc.

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

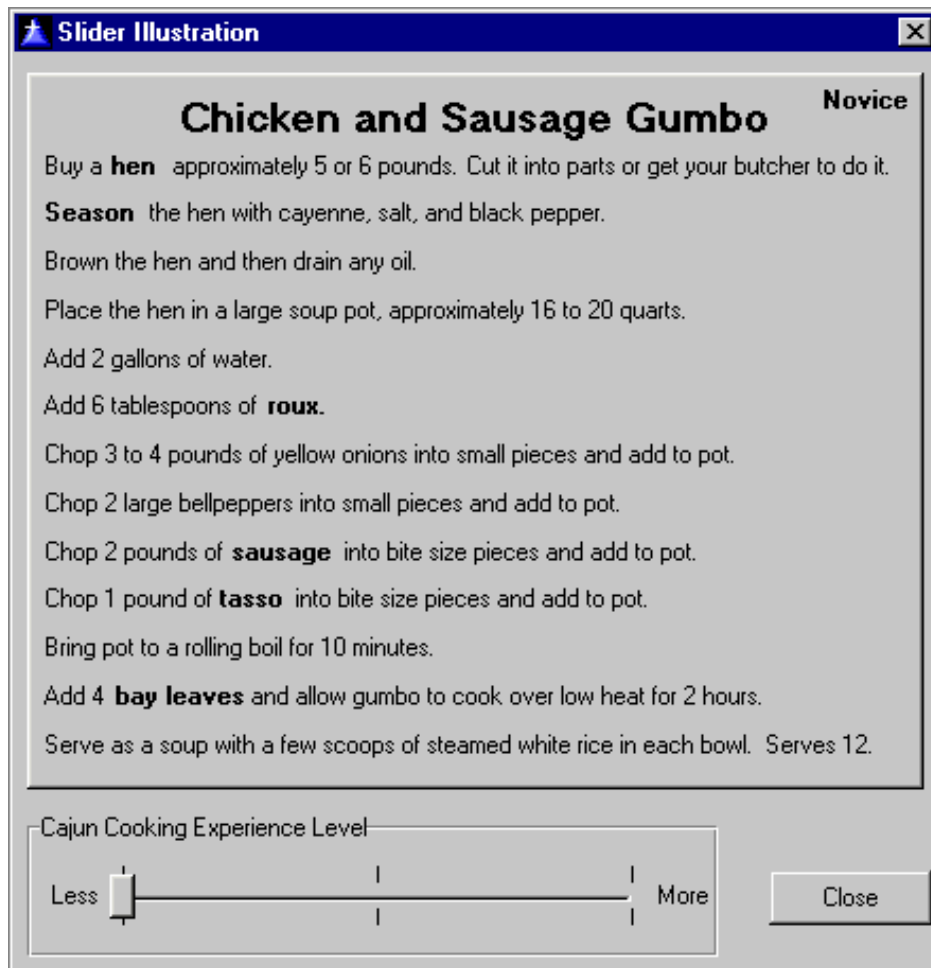
Main Page
Log In
Subscribe
Open Source
Links
Mailing Lists
Advertising
Submissions
Contact Us
Site Index
ClarionMag FAQ
Download PDFs
Search ClarionMag

easily adapted to different users? An interface is needed that can hide information but be easily adjusted for various skill levels. Sliders are perfect for this.

Jeff Slarve wrote an interesting slider class and template back when Clarion first began to support OOP. I took this software, enhanced it, and wrote a different template interface with some additional features.

Figure 1 shows the example application in all three modes.

Figure 1. The example application with the slider at each of the three settings.



**Slider Illustration** ✕

## Chicken and Sausage Gumbo Can Cook

Season a medium sized hen cut into parts with cayenne pepper, salt, and black pepper.  
Brown the hen and then drain the oil.  
Place hen in large soup pot along with 2 gallons of water.  
Add 6 tablespoons of roux.  
Add 3 to 4 pounds of chopped onion and 2 large chopped bellpeppers.  
Add 2 pounds of smoked sausage cut into bite size pieces.  
Add 1 pound of tasso cut into bite size pieces. Heavily smoked ham can be substituted.  
Bring water to a hard boil for 10 minutes.  
Turn heat down low, add 4 bay leaves, and allow to cook for 2 hours.  
Serve as a soup with a few scoops of white rice in each bowl. Serves 12.

Cajun Cooking Experience Level

Less  More Close

**Slider Illustration** ✕

## Chicken and Sausage Gumbo Cajun

Season and brown a hen and drain all of the oil.  
Add water and roux along with vegetables, tasso, sausage, and andouille.  
Bring to a boil for 10 minutes.  
Add a few bay leaves and cook on low fire for about 2 hours.

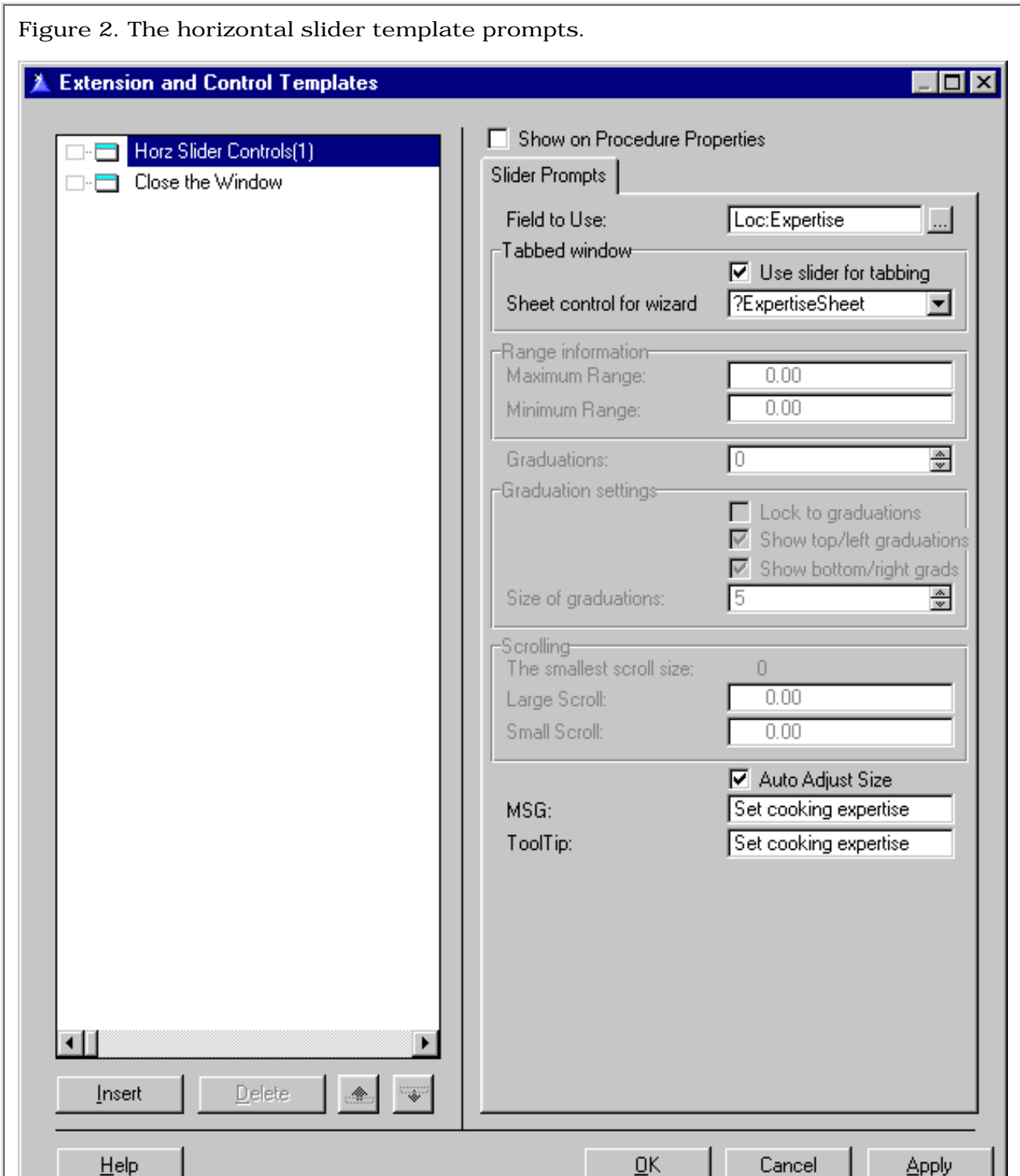
Cajun Cooking Experience Level

Less  More Close

To make this slider interface work first register the slider template (jsslid50.tpl). Then in your application create a form. On the form you need a sheet with the wizard attribute set, with tabs for each skill level of user. You may find it easiest to only set the wizard attribute when you're finished creating the information on each tab.

Then populate the slider control template on the window. In the case of the example app I'm including you'll see that I populated a horizontal slider (see Figure 2). The slider needs a field to keep track of the expertise level and I created a local data element called, imaginatively, `Loc:Expertise`. You can test for the value stored in this variable when you want to know what expertise level your end user is operating at. The next step is to check "Use Slider for Tabbing" and select the wizard sheet (`?ExpertiseSheet`) so the template knows which sheet contains the various expertise level tabs. That's all you need to do to add this functionality to your program.

Figure 2. The horizontal slider template prompts.



At runtime, moving the slider will select the different tabs. The experienced user can start at the most complex level or move the slider down to see the form in its simplest level. Or the slider will allow the average user to move up in the complexity of the form as s/he gains more experience and confidence.

How does this apply to your software development? One of the products that I have written and support processes information for consumer credit offices. In laymen's terms these are small loan companies. There are various levels of expertise in these offices. In some, you find a manager who has been loaning money for many years and who knows how every little detail of how a loan can affect a customer. On the opposite end, you may have a clerk who has zero experience in the field. The question each of them faces each day boils down to this:

If a customer borrows X amount of money for Y months, how much is his/her monthly payment?

The manager realizes that varying interest rates, different insurance coverages, and a variety of fees can all be manipulated to change this figure. But the inexperienced clerk has no idea how all of these things effect loans. The clerk isn't even allowed to give certain allowances that a manager can. So why clutter up the data entry screen for a clerk with the wealth of information and control that a manager needs?

This template handles that problem perfectly. The manager has a very detailed tab. The clerk has one with few details. As the clerk becomes more experienced and takes on more responsibility, he can move the slider in the direction of more detail. The manager, conversely, can decide he doesn't need all that complexity for every loan and start out at a less detailed position.

Of course, this is a simple illustration. In the real world, you would test for security levels of the end user (is the clerk allowed to change the interest rate?). But this template can even handle a complex form requiring multiple tabs. That can be easily implemented by placing sheets on each "expertise level" tab.

## Lagniappe

Lagniappe (lan – nyap') is a wonderful Cajun word. If you went shopping here years ago, the storekeeper nearly always gave you a little something once you made your purchase. A gentleman might receive a cigar. A lady might receive a spool of thread or a piece of lace. Lagniappe is that little something extra you get that you weren't expecting. So, in that tradition, here are [three bits of lagniappe](#).



## Thread Drift

And what would anything written by me be without thread drift?

I wrote an app a few years ago that was basically for point of sale. The industry leaders in this business were well known and dominated this DOS market for years. A customer that used these DOS packages approached me wanting a new custom package primarily because of some unique reporting requirements.

I wrote the app from a totally new and foreign viewpoint as I had no experience in this type of business. Because of this, the screens were very different from the packages the end users were familiar with.

There was a particular type of transaction that occurred approximately 200 times a day in each office. This transaction could have many subtle variations. Conceptually, the end user could go through 27 different prompts in order to complete the transaction. In standard DOS style, the other packages only required a newline to get through these prompts to accept the default. Let me say that a bit more clearly: the end user had to hit newline 27 times in the best case.

After sampling an entire month's business, I discovered the best case scenario occurred over 95% of the time.

I designed the entry screen for this type of transaction to:

1. ask for an account number
2. ask if this is a special case (newline meant no)

And that was all. If this was a special case, we went to the 27 prompt screen. If not, we completed the transaction in two steps.

Now, imagine the end users' reaction to this - especially when there were customers standing 10 deep at their counter. And imagine their customers' reaction when service time was cut so drastically.

As programmers, we're confronted with a variety of problems that we tend to standardize and normalize. Don't fall into that trap with your user interface.



First, you get a set of classes that illustrate using OOP in Clarion. This is an ABC independent implementation so you can use it in both ABC apps and Legacy apps. Next, you also get a template with some nice bells and whistles including full control of the graduation marks (size, number, lock to graduations) and range setting prompts so you can use the sliders in nearly any type of app.

And the final bit of lagniappe is an interesting Clarion app that illustrates using the slider for the "expertise level" interface. And it isn't just a boring data entry screen demonstrating an invoice or a loan agreement. It is an application that teaches you to cook gumbo assuming three different levels of experience in cooking. I hope you enjoy the application and the gumbo. Welcome to my kitchen!

[Download the source code.](#)

---

Andrew Guidroz II is an active member of the Topspeed community on the Internet and writes software for the insurance/finance industry as well as a host of other industries. His famous Cajun cookouts have become a central feature of Clarion conferences throughout the US. Andrew's Cajun website is <http://www.coonass.com>.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## Clarion News

April 26, 1999

### [TS Resources SQL Contest](#)

George Willbanks of TS Resources, Inc. is holding a contest for fast SQL logic using the Pervasive SQL language and a specified database (as announced at the Australian DevCon). The winner will receive a 10/100 hub, two PCI NICs and cables. See message #31878 in the Products.C4 newsgroup on tsnews.clarion.com for more details.

### [Clarion Jersey DevCovey '99 To Be Held May 22](#)

This mini-DevCon is a joint meeting of The Long Island, NY User Group, Clarion New Jersey Group and CW SuperGroup of Pennsylvania. Major topics of presentation and discussion are COOP (Clarion OOP), CW Project Management, Internet Connectivity, OLE/OCX implementation in CW and C projects in CW. Speakers include Mike Hanson, Phil Will, Pete Halsted, Troy Sorzano, Rob Cohan, and Waseem Naik.

### [Topspeed Closes Compuserve Forum](#)

Effective April 30, 1999 Topspeed will close its forum on Compuserve. Over half a million messages were posted in that forum throughout its life, but in recent months most of the activity has been on the newsgroups at tsnews.clarionmag.com.

April 19, 1999

### [NiceTouch Releases View Wizard](#)

Nice Touch Solutions has released View Wizard which facilitates on-demand custom list box formatting and browse sorting from within any application. Nice Touch also has a special wizard package price of \$649 which includes Query Wizard, Report Wizard, View Wizard, CrossTab Wizard and Spreadsheet Wizard.

### [CW Search Utility Beta](#)

Carl Barnes' CW Search Utility is now in public beta. This handy tool is specifically designed for searching Clarion code and sports a variety of formatting/viewing options. Carl has also released a patch to update CW Assistant to use the C5 DLLs. There are no new features in this release.

### [Clarion Profile Exchange Updated](#)

The bottom line: you use a program called Product Scope 32 Bookmarks to view product and company profiles that mostly highlight Clarion (TopSpeed) third party add-on companies and products.

### [NT Tools/Clarion Tips](#)

NT Tools software collects MS-SQL 6.5 Executive Log, NT Event Log and Registry data from NT Servers and Workstations throughout an entire network. Connections are not limited to a single network; all that's needed is an IP address and security access to the remote connection. All data gathering is done locally and stored in a central repository; there are no agents to run remotely.

### [Jersey DevCovey '99](#)

The New York, Pennsylvania and New Jersey Groups are pleased to

Great Opportunity!  
Salary to \$125,000+  
Pre-IPO Stock Opts  
Sunny Boca Raton

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

announce a joint meeting on May 22, 1999. Speakers include Mike Hanson, Eric Jakobowitz, Waseem Naik, Troy Sorzano, and Phil Will.

#### [NextAge Consulting Releases Imaging Templates](#)

NextAge Consulting has released the Imaging Templates for Clarion for Windows version 5 (ABC). The Imaging Templates are a full set of procedure and control templates, which enable you to include document storage functions in your application within ten minutes. The template works by integrating the imaging OCX's that ship as a free part of the Windows operating system and work with any TWAIN-compatible scanner. Documents can be stored as TIFF files on the hard drive or within your data files as BLOBS. The templates provide Imaging functions such as: viewing documents, thumbnails, scanning, printing, rotation, page control, and zooming as well as image path maintenance functions.

#### [L3 Technologies Updates dBASE & Paradox Drivers for C5](#)

L3 Technologies has updated its dBASE and native Paradox file drivers for C5. The Drivers are written using the TopSpeed File Driver Kit, ensuring complete compatibility with Clarion. Database access is achieved using the Borland Database Engine (BDE). This is the same engine that is used in the dBASE and Paradox Database systems, so full compatibility is ensured. The drivers can be purchased on-line and downloaded. Evaluation versions are available.

April 12, 1999

#### [Programmer's Power Tool Library](#)

LOGICBit creates hardware level multimedia tools (MMLib32) and has just announced the Clarion programmer's power tool library XLib32. XLib32 includes support for Text-to-Speech, TTY, TTY-to-Speech, 4 low level compression algorithms, 4 128 bit encryption algorithms and lot's more. Since LOGICBit tools are not templates you can rest assured that all LOGICBit tools work seamlessly in all version of windows.

#### [DevCon Latin America '99](#)

The second edition of the Clarion Developers Conference Latin America will be held May 13-15 in Buenos Aires, Argentina. Last year's conference had over 200 attendees, making this one of the largest Clarion DevCons in the world. Roy Rafalco, Topspeed's President and CEO will deliver the keynote address, and this year's focus is on training. For more information see the web site, email devcon@unisoft.com.ar, or phone +5411 4372-7243, fax +5411 4374-9469.

#### [Handy Tools Updated](#)

Handy Tools consist of a number of Clarion 5 templates and classes that extend or complement the normal functionality of the Clarion 5 application development environment. A free fully-working set of these template/classes and examples is available at the web site.

#### [Internet Framework Templates-HTTP Server Edition Released](#)

These general purpose templates allow intermediate-level developers to develop almost any kind of 32 bit HTTP server application desired. Includes examples of reading/writing/searching databases. Handles multiple connections and manages Windows socket calls.

April 6, 1999

#### [IceTips KB Updated](#)

The IceTips knowledgebase has been updated with new features, including a "limited results" page, category icons, Next/Previous links, soundex matches, statistics, help, and highlighting of search text. There are over 2400 bug reports, FAQs, and articles in the IceTips KB.

#### [Create Clarion CGI Programs With X-Works](#)

X-Works is a tool for web server application development - either new development or porting existing applications to WWW. It works with web server software on the server computer (not on client PC). It allows you

to write CGI scripts in VB, VFP, or any other languages on NT/95/98. X-Works is over 20 times faster than WinCGI. If you buy X-Works by May 31, 1999, you get a 20% discount (USD \$199 or \$559 instead of \$249 or \$699).

[Internet Framework Templates: HTTP Server Edition In Beta](#)

These templates will make it very easy for developers to quickly have HTTP server capability using Clarion. Developers will be free to develop almost any kind of HTTP server application they need. Pricing to be determined. Check out the What's New department at <http://www.logicentral.com> for Mar 29, 1999 and Feb 24, 1999.

[Clarion-Related Product Links Updated](#)

Actually this page is updated on a regular basis. If you haven't looked at our categorized list of Clarion links recently you might be surprised at what you can find!

---

[Read the March 1999 News](#)

Do you have a news story or press release we should know about? Send it to [editor@clarionmag.com](mailto:editor@clarionmag.com)

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## The Clarion Challenge

### The Clarion Code Challenge: A Compiler's Eye View

#### Part Two

By David Bayliss

(Editor's note: Some of the source code on this page exceeds the page boundaries when printed from a browser. You may wish to [download the April 26 PDF](#) which takes care of this problem.)

Recently Clarion magazine ran a [competition](#) to see who could produce the fastest and tightest code for a given problem. This is part two of David Bayliss's analysis of what the compiler is doing with that code. [Click here](#) to read part one.

Poul Jensen - 160 bytes

**NextAge**  
Consulting  
**Imaging**  
**Templates**  
Add Scanning and Document Storage to your app in 10 minutes  
**Only \$149**

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

```
PoulJensen Procedure(Long LongStart,Long LongEnd,Short TimeBlock,
    Byte RoundIndicator=False)
TimeToBill          LONG
Blocks              LONG
    code
    If Longstart > LongEnd
        TimeToBill = 8640001 - LongStart + LongEnd
    else
        TimeToBill = LongEnd - LongStart + 1
    End!if
    Blocks = Round((TimeToBill / TimeBlock/6000) ,1.0)
    If Not RoundIndicator and Blocks * TimeBlock |
        * 6000 + 1 < TimeToBill
        Blocks +=1
    End!if
    return(Blocks)
```

The first IF clause is a classic illustration of just how simple and efficient arithmetic is (in 32 bit) if it can be kept at long width.

```

!39 D8!          cmp eax,ebx          ! LongStart>LongEnd
!7E 0F!          jle 0FH
!BE 01 D6 83 00! mov esi,83D601H     ! ESI = 8640001
!29 C6!          sub esi,eax         ! ESI = 8640001-LongStart
!01 DE!          add esi,ebx         ! ESI = 8640001-LongStart+LongEnd
!89 74 24 10!    mov [esp][10H],esi ! Store in TimeToBill
!EB 07!          jmp 7
!29 C3!          sub ebx,eax         ! EBX = LongEnd-LongStart
!43!            inc ebx            ! EBX = LongEnd-LongStart+1
!89 5C 24 10!    mov [esp][10H],ebx ! Store in TimeToBill

```

Only 26 bytes of code in total, and two of those could have been saved by using a `CHOOSE!`

The `Blocks` computation is (by now) standard fare.

```

!8B 44 24 10!    mov eax,[esp][10H] ! TimeToBill into EAX
!E8 00 00 00 00! call Cla$DpushLong ! Onto stack
!0F BF F1!       movsx esi,cx       ! TimeBlock into ESI
!89 F0!          mov eax,esi        ! Then into eax
!E8 00 00 00 00! call Cla$DpushLong ! Then onto the stack
!E8 00 00 00 00! call Cla$DecDivide ! Divided
!B8 70 17 00 00! mov eax,1770H
!E8 00 00 00 00! call Cla$DPushLong
!E8 00 00 00 00! call Cla$DecDivide ! Divided by 6000
!B8 00 00 00 00! mov eax,$4
!B3 02!          mov bl,2
!66 B9 01 00!    mov cx,1
!E8 00 00 00 00! call Cla$DpushConstant ! Pushing 1.0 onto stack
!E8 00 00 00 00! call Cla$Dround    ! Rounding done
!E8 00 00 00 00! call Cla$DpopLong  ! Into EAX
!89 44 24 0C!    mov [esp][0CH],eax ! into Blocks.

```

The only strangeness is the move via `ESI` of `TimeBlock`. Did the compiler miss a trick? No it spotted one. It needed `TimeBlock` at 32 bit width later (for a multiplication) so it temp-stored the result in `esi`.

The `IF` clause is worth looking at for one reason. It shows the short-circuit computation of the `AND`. Look at the first two lines. If `RoundIndicator` (in `dl`) is non-zero then the rest of the `if` condition (including the multiplies) is not executed.

```

!80 FA 00!       cmp dl,0           ! IF RoundIndicator
!75 1A!          jne 1AH           ! Get out of here
!8B 44 24 0C!    mov eax,[esp][0CH] ! Load upBlocks
!0F AF C6!       imul eax,esi      ! TimeBlock at 32 bit width, stashed
earlier
!69 D8 70 17 00 00! imul ebx,eax,1770H ! * 6000 into EBX
!43!            inc ebx           ! Plus 1
!8B 44 24 10!    mov eax,[esp][10H]
!39 C3!          cmp ebx,eax       ! Compare with TimeToBill
!7D 04!          jge 4

```

Adding 1 to `Blocks` is so easy the compiler doesn't bother with a register.

```
!FF 44 24 0C!    inc dword [esp][0CH]
```

Finally move the result into `ax` to return.

```
!66 8B 44 24 0C!          mov ax,[esp][0CH]
```

## Summary

The code generated is actually quite reasonable and compact. The main time-hit is on the non-rounding case where Poul has still used the full rounding mechanism. He also has the double division problem that plagued a number of us. He could also get a slightly better result by rounding with ,1 rather than ,1.0. Again the variables could use the AUTO attribute.

Kurt Pawlikowski – 199 Bytes

```
KurtPawlikowski Procedure(Long loc:StartTime,Long loc:EndTime,
    Short loc:Units,Byte loc:Round=False)
loc:UnitsBilled    long ! Result number of units billed
code
if loc:StartTime > loc:EndTime then loc:EndTime += 8640000.
loc:UnitsBilled = int(((loc:EndTime - loc:StartTime)/|
    (loc:Units * 6000))) + (((loc:EndTime - loc:StartTime)|
    /(loc:Units * 3000))% 2) * loc:Round)
return(loc:UnitsBilled)
```

The first IF condition is handled cleanly; the use of += allows the compiler to go on using ebx for loc:EndTime which is nice.

```
!39 D8!          cmp eax,ebx
!7E 06!          jle 6
!81 C3 00 D6 83 00!  add ebx,83D600H
```

Now comes the monster expression. First the compiler performs the subtraction leaving the result in ebx.

```
!29 C3!          sub ebx,eax
```

The result is now moved into eax prior to moving onto the decimal stack to perform the division. The INT function is also performed on the decimal stack where the result is left whilst the right hand side of the addition is tackled.

```
!89 D8!          mov eax,ebx
!E8 00 00 00 00!  call Cla$DpushLong ! End-Start onto stack
!0F BF C9!          movsx ecx,cx        ! 32 bit loc:Units into ECX for later
!69 C1 70 17 00 00!  imul eax,ecx,1770H ! loc:Units * 6000 into EAX
!E8 00 00 00 00!  call Cla$DpushLong ! Onto stack
!E8 00 00 00 00!  call Cla$DecDivide ! Divided
!E8 00 00 00 00!  call Cla$Dint      ! INT(Value) left on decimal stack
```

Starting inside the brackets:

```
!89 D8!          mov eax,ebx        ! End-Start already in EBX
!E8 00 00 00 00!  call Cla$DpushLong ! Now onto decimal stack
!69 C1 B8 0B 00 00!  imul eax,ecx,0BB8H ! Multiply loc:Units by 3000
!E8 00 00 00 00!  call Cla$DpushLong ! Onto decimal stack
!E8 00 00 00 00!  call Cla$DecDivide ! Perform division
```

Now the trouble starts. A DECIMAL modulus of a LONG is computed at REAL width, so the % is much more expensive than Brian's case.

```

!E8 00 00 00 00!      call Cla$DpopReal      ! Decimal result into st0
!DD 5C 24 24!        fstp qword [esp][24H],st0 ! Temp store
!83 EC 08!           sub esp,8              ! Prepare to pass real
parameter
!DD 44 24 2C!        fld st0,qword [esp][2CH] ! Load from temp (esp has
changed!)
!DD 1C 24!           fstp qword [esp][0],st0  ! Pass real parameter
!83 EC 08!           sub esp,8              ! Prepare for second real
parameter
!DD 05 00 00 00 00!  fld st0,qword [dword $3] ! Load constant 2
!DD 1C 24!           fstp qword [esp][0],st0  ! Pass real parameter
!E8 00 00 00 00!      call Cla$modulus       ! Real modulus operation
!DD 5C 24 1C!        fstp qword [esp][1CH],st0 ! Store result

```

Now the compiler has the left hand side of the addition on the decimal stack and the left hand side stored as a REAL in memory. The addition now has to be performed at REAL width. Panic time.

Get the original Dint result into memory as a REAL; then it's a controlled situation.

```

!E8 00 00 00 00!      call Cla$DPopReal
!DD 5C 24 14!        fstp qword [esp][14H],st0

```

The compiler now has everything as REALS so can start doing the floating point math, which it does quite well.

```

!DD 44 24 14!        fld st0,qword [esp][14H] ! Load the Dint result onto
the FPU for later
!DD 44 24 1C!        fld st0,qword [esp][1CH] ! Load the modulus result
onto the chip
!0F B6 C2!          movzx eax,dl           ! Get loc:Round into eax
!89 44 24 10!       mov [esp][10H],eax    ! Into memory
!DB 44 24 10!       fld st0,dword [esp][10H] ! Then onto the FPU
!DE C9!            fmulp st1,st0      ! Perform the multiplication
!DE C1!            faddp st1,st0     ! Then the addition

```

Finally the result has to come off of the FPU and into eax so that it can be returned as a SHORT. (See Jeff Slarve's code.)

```

!D9 7C 24 0A!       fstcw [esp][0AH]
!D9 7C 24 08!       fstcw [esp][8]
!66 81 4C 24 08 3F 0C! or word [esp][8],0C3FH
!D9 6C 24 08!       fldcw [esp][8]
!DB 5C 24 0C!       fistp dword [esp][0CH],st0
!DB E2!            fclex
!D9 6C 24 0A!       fldcw [esp][0AH]
!8B 44 24 0C!       mov eax,[esp][0CH]
!89 44 24 2C!       mov [esp][2CH],eax
!66 8B 44 24 2C!   mov ax,[esp][2CH]

```

## Summary

This is one of those cases where the Clarion compiler getting the answer "right" can lead to code you weren't expecting. The % operation is best defined between longs (see Brian Staff's code). Once you are into DECIMALS/REALS it becomes quite a complex operation.

Gordon Smith – 262 Bytes



```
GordonSmith Procedure(Long b,Long e,Short u,Byte r=False)
a real, auto
code
a = ((e - b + 8640000) % 8640000) / (u * 6000)
return CHOOSE(r, round(a, 1), a + CHOOSE(a - int(a) > 0, 1, 0))
```

Like Brian, Gordon has managed to keep his modulus calculation at integer width, which the compiler inlines. This produces quite a lot of code but most of the instructions are fast.

```
!29 F0!          sub eax,esi          ! EAX = e - b
!05 00 D6 83 00!  add eax,83D600H     ! EAX = inner parenthesis
!BE 00 D6 83 00!  mov esi,83D600H     ! Right hand operand of modulus
!99!            cdq
!F7 FE!          idiv esi            ! Perform modulus operation
!89 D0!          mov eax,edx
!89 C2!          mov edx,eax
!81 F2 00 D6 83 00!  xor edx,83D600H
!7D 09!          jge 9              ! 'Wiggle' to obey Clarion semantics.
!85 C0!          test eax,eax
!74 05!          je 5
!05 00 D6 83 00!  add eax,83D600H
```

Having computed the modulus the compiler then places the result on the decimal stack ready for the division. The  $u*6000$  is again inline assembler.

```
!E8 00 00 00 00!  call Cla$DpushLong  ! Left hand onto decimal stack
!0F BF C1!          movsx eax,cx         ! EAX = u (at 32 bit width)
!69 C0 70 17 00 00!  imul eax,eax,1770H ! u * 6000
!E8 00 00 00 00!  call Cla$DpushLong  ! Onto decimal stack
!E8 00 00 00 00!  call Cla$DecDivide  ! Perform division
```

Now Gordon stores the result in a REAL (similar to Jeff's code).

```
!E8 00 00 00 00!  call Cla$DpopReal   ! Decimal into st0
!DD 5C 24 38!          fstp qword [esp][38H],st0 ! Intermediate result stored
(for use in INT(A))
!DD 44 24 38!          fld st0,qword [esp][38H] ! Re-loaded
!DD 5C 24 30!          fstp qword [esp][30H],st0 ! Moved into a
```

The compiler then chooses which arm of the outer CHOOSE to evaluate. Interestingly Gordon is using an enumerated CHOOSE ( the variable is used as a number ) rather than a logical one (  $r <> 0$  ). So if  $r$  was two this would go down the second arm.

```
!80 FB 01!          cmp bl,1            ! Compare r against 1
!74 4C!             je 4CH              ! If not one go down the 'else' (second) arm.
```

The "else" (second) arm starts with the compiler evaluating the inner CHOOSE. first the subtraction:

```

!83 EC 08!          sub esp,8
!DD 44 24 38!      fld st0,qword [esp][38H]      !Load up copy of a
!DD 1C 24!         fstp qword [esp][0],st0        !Store as parameter
!E8 00 00 00 00!   call Cla$INT                    ! Call INT function
!DD 5C 24 28!      fstp qword [esp][28H],st0      ! Store result
!DD 44 24 30!      fld st0,qword [esp][30H]      ! Load up a
!DD 44 24 28!      fld st0,qword [esp][28H]      ! Load up INT(A)
!DE E9!           fsubp st1,st0                ! Perform a-INT(A)

```

Then the result of the subtraction is compared against zero and the value of the inner CHOOSE is computed. Note that the result of the inner CHOOSE is LONG even though the logical expression involves REALS. That is because the three parameters are of type LOGICAL, LONG, LONG.

```

!DD 05 00 00 00 00!   fld st0,qword [dword $2]      ! Load up zero
!DE D9!              fcompp st0,st1                ! Compare with a-INT(a)
!DF E0!              fstsw ax                    ! Status into ax
!F6 C4 01!           test ah,1                      ! Compute (integral) CHOOSE
result
!74 07!              je 7
!B8 01 00 00 00!     mov eax,1
!EB 05!              jmp 5
!B8 00 00 00 00!     mov eax,0

```

Now the value of the CHOOSE (in eax) has to be cast up to a real so that it can be added to a.

```

!DD 44 24 30!      fld st0,qword [esp][30H]      ! Re-load a
!89 44 24 24!      mov [esp][24H],eax        ! Move EAX into memory
!DB 44 24 24!      fld st0,dword [esp][24H]      ! Load onto fpu
!DE C1!           faddp st1,st0                ! Perform addition

```

The rounding arm is exactly the same as Jeff's as is the casting code to get the REAL result back into ax to return it.

### Summary

This code is extremely similar to Jeff's. I suspect that the main reason for the speed difference is that Gordon has avoided the second division in the first line.

Alan Telford – 204 Bytes

```

AlanTelford Function(Long p:StartTime, Long p:EndTime,
    Short p:BlockLength, Byte p:UseRounding)
! Return the number of blocks of time (of p:BlockLength minutes)
! which occur between p:StartTime and p:EndTime
! IF not rounding, then charge for each full or any part segment
! If rounding, then only charge for each full and any part segments
! which are as long as the block length
l:BlockCount Short ! return value from function
l:Minutes Short ! intermediate value which holds minutes
!between starttime and endtime
eMidnight Equate(8640000)
CODE
! If endtime is after midnight then add 24 hours so
! calculation comes out correct
IF p:EndTime < p:StartTime
    p:EndTime += eMidnight
END
l:Minutes = (p:EndTime - p:StartTime) / 6000
l:BlockCount = l:Minutes / p:BlockLength

```

```

IF ~p:UseRounding and l:Minutes % p:BlockLength |
OR l:Minutes % p:BlockLength >= p:BlockLength / 2
  l:BlockCount += 1
END
return l:BlockCount

```

I must admit that when I saw Alan's entry I grinned. He and John are the only two to have produced what I would call "proper" code: the kind of thing you would expect to get inside an insurance company. I looks as if it ought to be very safe, very methodical, and very inefficient.

One of the great things about the Clarion compiler is that all of this "properness" gets optimized out and a result is produced that can easily hold its head up against the hot-n-heavy hackers.

The opening IF statement is crunched into 9 bytes.



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

```

!39 F0!          cmp eax,esi          ! Compare p:EndTime v's p:StartTime
!7D 05!          jge 5
!05 00 D6 83 00! add eax,83D600H      ! Add eMidNight to p:EndTime if
required

```

The l:Minutes computation is a standard usage of the decimal stack for a division.

```

!29 F0!          sub eax,esi          ! EAX is EndTime-StartTime
!E8 00 00 00 00! call Cla$DpushLong  ! Onto Stack
!B8 70 17 00 00! mov eax,1770H
!E8 00 00 00 00! call Cla$DpushLong  ! 6000 onto stack
!E8 00 00 00 00! call Cla$DecDivide  ! Divide
!E8 00 00 00 00! call Cla$DpopLong   ! Result into eax
!66 89 44 24 10! mov [esp][10H],ax   ! Store in l:Minutes

```

The BlockCount computation follows a similar pattern :

```

!0F BF 5C 24 10! movsx ebx,word [esp][10H] ! l:Minutes into ebx at 32
bit width
!89 D8!          mov eax,ebx          ! EBX has copy for later %
computation
!E8 00 00 00 00! call Cla$DpushLong  ! Then onto stack
!0F BF C9!          movsx ecx,cx         ! Sign extend p:BlockLength (for
later)
!89 C8!          mov eax,ecx
!E8 00 00 00 00! call Cla$DpushLong  ! Onto Stack
!E8 00 00 00 00! call Cla$DecDivide  ! Division
!E8 00 00 00 00! call Cla$DpopLong   !
!66 89 44 24 12! mov [esp][12H],ax   ! Result into BlockCount

```

The IF statement gets very tangled in the assembler mainly because of the need to guarantee short-circuit evaluation. (The expression finishes evaluating as soon as the result is known). Evaluation is always left to right although ANDs bind tighter than ORs. So first a check is made on UseRounding.

```

!80 FA 00!          cmp dl,0
!75 16!          jne 16H

```

If rounding is on then the first modulus has to be computed. This is integral and so is inlined.

```

!89 D8!          mov eax,ebx ! EAX set to L:Minutes (remembered
previously)
!99!            cdq
!F7 F9!          idiv ecx    ! ECX holds p:BlockLength (again computed
previously)
!89 D0!          mov eax,edx
!31 C8!          xor eax,ecx
!7D 06!          jge 6      ! Usual modulus wiggling.
!85 D2!          test edx,edx
!74 02!          je 2
!01 CA!          add edx,ecx

```

The modulus result (in `edx`) then has to be checked against zero.

```

!83 FA 00!       cmp edx,0
!75 38!          jne 38H

```

If `eax` has a value then the body of the `IF` is executed. Otherwise the `OR` condition has to be checked as follows :

First the right hand side of the `>=` is computed. This may seem a little odd, but both sides of the expression have to be evaluated and the language does not specify which one is computed first (this is true for all operators except `AND` and `OR`).

```

!89 C8!          mov eax,ecx    !P:Blocklength into EAX
!E8 00 00 00 00! call Cla$DpushLong !Onto stack
!B8 02 00 00 00! mov eax,2
!E8 00 00 00 00! call Cla$DpushLong !Two onto stack
!E8 00 00 00 00! call Cla$DecDivide !Division performed

```

This result is left on the decimal stack whilst the modulus is computed. Again this is integral and can be inlined.

```

!89 D8!          mov eax,ebx    ! L:Minutes into eax
!99!            cdq
!F7 F9!          idiv ecx    ! p:Blocklength already around
!89 D0!          mov eax,edx
!89 C3!          mov ebx,eax
!31 CB!          xor ebx,ecx    ! Standard modulus wiggle.
!7D 06!          jge 6
!85 C0!          test eax,eax
!74 02!          je 2
!01 C8!          add eax,ecx    ! Result into EAX

```

Now this is an interesting part. The right hand side of this expression is a `DECIMAL`, and the left is a `LONG`. The `>=` therefore has to be computed at decimal width, and this is done via the decimal stack.

```

!E8 00 00 00 00! call Cla$DpushLong    ! EAX onto decimal stack
!E8 00 00 00 00! call Cla$DecDistinctR ! Compare top two items on
stack, result in EAX
!83 F8 00!       cmp eax,0
!7C 05!          jl 5

```

After all that hard work computing the `IF` condition the actual body of the `IF` clause is so trivial the compiler doesn't even use a register.

```

!66 FF 44 24 12! inc word [esp][12H] ! p:BlockCount += 1

```

The result is then returned in ax.

```
!66 8B 44 24 12!          mov ax,[esp][12H]
```

## Summary

Alan's code is a beautiful example of how straightforward do-it-as-you-see-it code can actually result in a very compact and acceptable result.

In terms of improvement there are a few things that can be done. First is the use of the `AUTO` attribute on the variables. The compiler has also had to use `MOVSX` a couple of times to extend a 16 bit value up to 32 bit. Generally in 32 bit systems it is better to use `LONG` than `SHORT` (although Alan would do better in 16 bit).

I suspect the main sloth culprit (shared with a number of people, including myself) is the "second division" problem.

Alan has also tripped slightly with the `l:Minutes % p:BlockLength` computation which appears down both arms of the `OR` clause in the `IF`. You might reasonably expect this expression to be spotted by the compiler as a common sub-expression (as it did for Brian a few times) and only computed once. Unfortunately it cannot do this because of the guarantee that logical expressions are short-circuit evaluated.

Peter Gysegem A- 168 Bytes

```
PeterGysegemA      PROCEDURE (LONG StartDate, LONG StartTime,
                          LONG EndDate, LONG EndTime, SHORT Increment, BYTE RoundFlag)

Elapsed  LONG      !Elapsed time in 100ths of a second
Ticks   LONG      !Billing increment in 100ths of a second
Units   SHORT     !Number of billing units

CODE
IF StartDate = EndDate
    Elapsed = EndTime - StartTime
ELSE
    Elapsed = EndTime + 8640000 - StartTime
END
Ticks = Increment * 6000

IF RoundFlag
    Units = ROUND(Elapsed/Ticks, 1)
ELSE
    Units = Elapsed/Ticks
END

RETURN(Units)
```

This is the fastest code on the block and when you look at the disassembly you can see why.

```
!39 C8!          cmp eax,ecx          ! Compare StartDate with EndDate
!75 08!          jne 8
!29 DA!          sub edx,ebx          ! EndTime-StartTime
!89 54 24 0C!    mov [esp][0CH],edx  ! Result into Elapsed
!EB 0C!          jmp 0CH
!81 C2 00 D6 83 00! add edx,83D600H     ! EndTime + 8640000
!29 DA!          sub edx,ebx          ! EndTime + 8640000 - StartTime
!89 54 24 0C!    mov [esp][0CH],edx  ! Into Elapsed
```

The computation of `Ticks` is also straightforward. The `Increment` variable comes in on the stack because after four parameters the compiler has run out of registers to pass parameters around.

```

!0F BF 44 24 18!      movsx eax,word [esp][18H]  ! EAX = Increment
!69 C0 70 17 00 00!   imul eax,eax,1770H       ! EAX = Increment * 6000
!89 44 24 08!        mov [esp][8],eax         ! Into Ticks

```

RoundFlag is also on the stack although as it is only tested it doesn't matter as the compiler can use it from memory.

```

!80 7C 24 14 00!      cmp byte [esp][14H],0    ! IF RoundFlag
!74 32!               je 32H

```

The reason Peter has such a fast routine is that up until this point he has stayed entirely with integer arithmetic. The Units computation involves a division and thus goes via the decimal stack although even here the code is straightforward. This is the ROUNDING case; the unrounded is the same but without the call to Dround.

```

!8B 44 24 0C!        mov eax,[esp][0CH]      ! EAX = Elapsed
!E8 00 00 00 00!     call Cla$DpushLong     ! Onto Stack
!8B 44 24 08!        mov eax,[esp][8]       ! EAX = Ticks
!E8 00 00 00 00!     call Cla$DpushLong     ! Onto stack
!E8 00 00 00 00!     call Cla$DecDivide     ! Divide
!B8 01 00 00 00!     mov eax,1
!E8 00 00 00 00!     call Cla$DpushLong     ! 1 Onto Stack
!E8 00 00 00 00!     call Cla$Dround       ! Perform rounding
!E8 00 00 00 00!     call Cla$DpopLong     ! Result into EAX
!66 89 44 24 06!     mov [esp][6],ax       ! Into Units

```

## Summary

Peters code is very similar to mine except he (correctly) avoided the double division. There are still a few tweaks available. AUTO is one. I would also be tempted to try RETURNING the result rather than going via Units. I think he could have got a little more speed by avoiding the two date parameters as well.

John Cunningham – 304 Bytes

```

JohnCunningham Procedure(Long TIM:timeStarted,Long TIM:TimeEnded,
    Short TIM:TimePeriod,Byte TIM:RoundedPeriod=False)
TIM:MinutesWorked    long
TIM:PeriodsWorked    short
code
Do CalcMinutes
IF TIM:MinutesWorked % TIM:TimePeriod <> 0
! Left over minutes
TIM:PeriodsWorked += 1
IF TIM:RoundedPeriod
! Rounded
IF (TIM:MinutesWorked % TIM:TimePeriod) < (TIM:TimePeriod / 2)
! Minimum of 1 period billed
IF TIM:PeriodsWorked > 1
TIM:PeriodsWorked -= 1
END
END
END
END
return(TIM:PeriodsWorked)

CalcMinutes Routine
IF TIM:TimeEnded >= TIM:TimeStarted
! Before Midnight

```

```

TIM:MinutesWorked = ((TIM:TimeEnded-1)-(TIM:TimeStarted-1)+1)/6000
ELSE
! After Midnight
TIM:MinutesWorked = (((8640001-TIM:TimeStarted-1)+|
(TIM:TimeEnded-1))+1)/6000
END
TIM:PeriodsWorked = INT(TIM:MinutesWorked/TIM:TimePeriod)

```

Without doubt the most interesting feature of Johns code (from the perspective of this analysis) is the use of a routine. One a procedure has a routine it is a parent, and as we all know parent-hood is the time when we have to stop being fast and flash and start being responsible. So at the head of Johns procedure is this code:

```

!89 45 F4!          mov [ebp*1][-0CH],eax
!89 5D F8!          mov [ebp*1][-8],ebx
!66 89 4D EC!       mov [ebp*1][-14H],cx

```

What is happening is that all of the variables the child (routine) wants have to be put in memory where the child can get at them. Also whenever the parent wants the value it has to go to memory too (in case the child changed it). These variables are described as threatened, which essentially means they must always be used from memory.

The code for the routine is then fairly standard except that many of the variables used have to be fetched via an implicit pointer to the parent. So the coding for the IF becomes :

```

!8B 5D FC!          mov ebx,[ebp*1][-4]          ! Make EBX 'pointer to
procedure variables'
!8B 43 F4!          mov eax,[ebx*1][-0CH]       ! Load up TimeStarted
!39 43 F8!          cmp [ebx*1][-8],eax         ! Compare to TimeEnded

```

The compiler does spot the TimeEnded-1 appearing in both arms of the IF and precomputes it :

```

!9C!               pushfd
!8B 4B F8!          mov ecx,[ebx*1][-8]         ! Load TimeEnded from parent
!49!               dec ecx                    ! TimeEnded-1
!9D!               popfd

```

The two MinutesWorked computations are similar to things I've explained earlier so I'll just annotate the assembler for the simpler one.

```

!89 C8!            mov eax,ecx                 ! EAX = TimeEnded-1
(precomputed above)
!8B 4B F4!          mov ecx,[ebx*1][-0CH]       ! ECX = TimeStarted
!49!               dec ecx                    ! ECX = TimeStarted-1
!29 C8!            sub eax,ecx                 ! EAX =
(TimeEnded-1)-(TimeStarted-1)
!40!               inc eax                    ! EAX =
(TimeEnded-1)-(TimeStarted-1)+1
!E8 00 00 00 00!   call Cla$DpushLong         ! Onto decimal stack
!B8 70 17 00 00!   mov eax,1770H
!E8 00 00 00 00!   call Cla$DpushLong         ! 6000 => Decimal stack
!E8 00 00 00 00!   call Cla$DecDivide         ! Division
!E8 00 00 00 00!   call Cla$DpopLong         ! Result Into EAX
!89 43 F0!          mov [ebx*1][-10H],eax      ! Into MinutesWorked.

```

Finally within the routine the compiler performs the division and then the INT. INT on a DECIMAL is done on the decimal stack and is extremely cheap.

```

!8B 43 F0!          mov eax,[ebx*1][-10H]      ! MinutesWorked
!E8 00 00 00 00!    call Cla$DpushLong        ! Onto decimal stack
!0F BF 43 EC!       movsx eax,word [ebx*1][-14H]! TimePeriod
!E8 00 00 00 00!    call Cla$DpushLong        ! Onto decimal stack
!E8 00 00 00 00!    call Cla$DecDivide        ! Divided
!E8 00 00 00 00!    call Cla$Dint             ! Int Function
!E8 00 00 00 00!    call Cla$DpopLong        ! Result Into EAX
!66 89 43 EE!       mov [ebx*1][-12H],ax      ! And into PeriodsWorked (in
parent)

```

The main body of the code is then dominated by the two modulus operations, both of which are inlined. Note the compiler has to keep going back to memory for the threatened variables.

This is the first IF condition:

```

!8B 45 F0!          mov eax,[ebp*1][-10H]      ! EAX = Minutes worked
!0F BF 4D EC!       movsx ecx,word [ebp*1][-14H]! ECX = TimePeriod
!99!               cdq
!F7 F9!           idiv ecx                  ! Modulus operation
!89 D0!           mov eax,edx
!31 C8!           xor eax,ecx
!7D 06!           jge 6                      ! Modulus wiggle
!85 D2!           test edx,edx
!74 02!           je 2
!01 CA!           add edx,ecx
!83 FA 00!        cmp edx,0                  ! Test result against zero
!74 4D!           je 4DH                     ! Jump out if 0.

```

Incrementing PeriodsWorked can be done in memory:

```
!66 FF 45 EE!      inc word [ebp*1][-12H]
```

Next RoundedPeriod is checked. This variable is not threatened so the compiler has kept it in registers (bl).

```
!80 FB 00!        cmp bl,0
!74 44!           je 44H
```

The second modulus computation and corresponding IF condition is exactly the same as Alan Telford's and similar code is generated.

```

!89 C8!           mov eax,ecx                ! EAX = TimePeriod
!E8 00 00 00 00!    call Cla$DpushLong        ! Onto decimal stack
!B8 02 00 00 00!    mov eax,2
!E8 00 00 00 00!    call Cla$DpushLong        ! 2 onto decimal stack
!E8 00 00 00 00!    call Cla$DecDivide        ! Division, leaving result on
stack
!8B 45 F0!          mov eax,[ebp*1][-10H]      ! EAX = Minutes worked
!99!               cdq
!F7 F9!           idiv ecx                  ! Divide by TimePeriod
!89 D0!           mov eax,edx
!89 C3!           mov ebx,eax
!31 CB!           xor ebx,ecx                ! Wiggle
!7D 06!           jge 6
!85 C0!           test eax,eax
!74 02!           je 2
!01 C8!           add eax,ecx
!E8 00 00 00 00!    call Cla$DpushLong        ! Result onto stack
!E8 00 00 00 00!    call Cla$DecDistinctR     ! Compare top two items on

```



```

decimal stack
!83 F8 00!           cmp eax,0           ! A zero return implies
equality
!7D 0B!             jge 0BH
!66 83 7D EE 01!    cmp word [ebp*1][-12H],1 ! Finally PeriodsWorked is
checked against 1
!7E 04!             jle 4
!66 FF 4D EE!       dec word [ebp*1][-12H] ! And decremented if greater
!66 8B 45 EE!       mov ax,[ebp*1][-12H] ! Result into AX to return.

```

## Summary

Johns code was very similar to a number of other entries. It eventually ran slower because of the extra work the compiler did sharing the variables between parent procedure and child routine. The comments in the other summaries apply (AUTO, double division etc), but the main one here is that in general you should try to avoid sharing variables between procedures and routines. You can do it (and it works) but here an efficiency penalty of about 20% has been paid.

## Carl Barnes – 198 Bytes

```

CarlBarnes Procedure(Long StartTime,Long StopTime,Short Period,
Byte Roundit=False)

```

```

CODE
!ASSERT(Period = 0 )
!ASSERT(RoundIt > 1)
RETURN INT((((StopTime - StartTime) / 6000 + 1440 ) % 1440 ) |
/ Period + RoundIt / 2 )

```

Given the two ASSERTs are commented out (check out the use of ? with ASSERT) this routine shows just how much code you can get from one relatively innocent looking line of code.

The compiler starts by working inside-out. First the subtraction :

```

!29 F0!           sub eax,esi           ! EAX = StopTime-StartTime

```

Then the division by 6000 :

```

!E8 00 00 00 00!   call Cla$DpushLong   ! EAX onto decimal stack
!B8 70 17 00 00!   mov eax,1770H        ! EAX = 6000
!E8 00 00 00 00!   call Cla$DpushLong   ! onto decimal stack
!E8 00 00 00 00!   call Cla$DecDivide   ! Division performed

```

Now the addition of 1440, the result of the division is a decimal so the addition has to be performed at decimal width:

```

!B8 A0 05 00 00!   mov eax,5A0H         ! EAX = 1440
!E8 00 00 00 00!   call Cla$DpushLong   ! onto Decimal stack
!E8 00 00 00 00!   call Cla$DecAdd      ! Addition result on decimal stack

```

The compiler now has to perform a modulus operation upon the decimal result. As with Kurt's code this becomes a floating point operation:

```

!E8 00 00 00 00!      call Cla$DpopReal      ! Result on decimal stack
needs to become real
!DD 5C 24 28!        fstp qword [esp][28H],st0 ! Result Temp-stored
!83 EC 08!           sub esp,8              ! Prepare to pass real
parameter
!DD 44 24 30!        fld st0,qword [esp][30H] ! Load up temp-stored result
!DD 1C 24!           fstp qword [esp][0],st0  ! Pass parameter
!83 EC 08!           sub esp,8              !
!DD 05 00 00 00 00! fld st0,qword [dword $0] !
!DD 1C 24!           fstp qword [esp][0],st0  ! Pass Period as real
parameter
!E8 00 00 00 00!      call Cla$modulus       ! Modulus operation
!DD 5C 24 20!        fstp qword [esp][20H],st0 ! Result stored away **1

```

The compiler down computes the right hand side of the outer addition. Because the left hand side is a REAL value the right hand side has to be computed at REAL width too:

```

!0F B6 C2!           movzx eax,dl           ! RoundIt into EAX
!E8 00 00 00 00!      call Cla$DpushLong     ! Onto decimal stack
!B8 02 00 00 00!      mov eax,2              !
!E8 00 00 00 00!      call Cla$DpushLong     ! 2 onto stack
!E8 00 00 00 00!      call Cla$DecDivide     ! Divided
!E8 00 00 00 00!      call Cla$DpopReal     ! Result as real
!DD 5C 24 18!        fstp qword [esp][18H],st0 ! Stored away **2

```

The compiler now returns to the left hand side of the addition where it still has to perform the division by Period. The left hand side is up at REAL width at this point so the division is done using floating point math:

```

!83 EC 08!           sub esp,8              !
!DD 44 24 28!        fld st0,qword [esp][28H] ! Load result stored away at
**1
!0F BF C1!           movsx eax,cx           ! EAX = Period
!89 44 24 1C!        mov [esp][1CH],eax     ! Stored into memory
!DB 44 24 1C!        fild st0,dword [esp][1CH] ! To load into FPU
!DE F9!             fdivp st1,st0         ! Division performed

```

Having finally got both sides of the addition the addition can happen:

```

!DD 44 24 20!        fld st0,qword [esp][20H] ! Load result temp-stored at
**2
!DE C1!             faddp st1,st0         ! Add to left hand side
(already on chip)
!DD 1C 24!           fstp qword [esp][0],st0  ! Store the result as a
parameter
!E8 00 00 00 00!      call Cla$INT           ! And pass to INT routine

```

Having the result in St0 the compiler then goes through the usual rigmarole of getting it into `eax` (see Jeff Slarve's code).

## Summary

This routine really demonstrates one of the gotchas (in terms of efficiency) of Clarions automated type conversion. Essentially, within an expression, the width of a computation nearly always gets wider. In many ways this is good (Clarion guarantees to get the result correct, many languages don't) but it does mean that the compiler/library may be working far harder to achieve a given result than the code you typed may lead you to expect.

---

[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.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## The SQL Answer Cowboy

Andy "Cowboy" Stapleton is the acknowledged Clarion SQL guru and a regular presenter at Clarion conferences around the world. His company, [Cowboy Computing Solutions](#), produces SQL templates and classes for Clarion.

[Click here](#) to submit your SQL question to Andy.

Tom Foley: I understand that TS no longer recommend the use of {Prop:SQL}. What is the replacement and can you explain the rationale?

Cowboy: To my knowledge {Prop:SQL} is not going away but you have to be careful using it, which is why they don't recommend it. If you use {Prop:SQL} and NEXT() with a PUT() you must insure that you have the primary key fields in the SELECT statement.

The reason is that PUT and NEXT use the primary fields to insure the update of the record is accurate. If you don't provide this in the SELECT Statement both the PUT and NEXT will not be aware of the record position and subsequently give errant results or none at all. You can get around not using the primary in the Select but you will have to create your own Update Statements.

TS is redeveloping the file drivers for additional functionality, but I cannot discuss this yet.

Yeoh Eng Loke: I have read your seminar materials for Devcon 98 where a sample file was downloadable. In that particular file called 'Devcon.doc', all the examples of stored procedures and triggers are shown using SQL Anywhere. Do you have an equivalent in MS-SQL?

Cowboy: We are currently working on an MS-SQL project and from this we will strip out examples of these items with documentation. I have also entered into an agreement with George Willbanks [TS Resources Inc] to produce a set of CD-ROM training items. Between the example programs in the final release of the CCS Templates and the CD-Roms there should be quite a bit of information about these these database flavors.

The syntax of MS-SQL is different from the syntax of Sybase but there are similarities. Here is an example of MS-SQL trigger:

```
[Trigger Definition ]
Trigger checks the format type in the setup table to see
if the user wants First then Last or Last then First,
Updates the FULLNAME field to match.
```

```
CREATE TRIGGER Names_FullName ON Names
FOR INSERT, UPDATE
AS
DECLARE @NameFormat varchar(10);
SELECT @NameFormat = NameFormat
FROM SETUP WHERE setupsysid = 1;
IF UPDATE(Fname) OR UPDATE(Lname)
```



etc  
2000

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

**NextAge**  
Consulting  
**Imaging**  
**Templates**

Add Scanning and  
Document Storage  
to your app  
in 10 minutes

**Only \$149**

```

IF @NameFormat = 'FirstLast'
    UPDATE NAMES SET FullName = Fname + ' ' + Lname
    WHERE Names.NameSysID =
        ANY(SELECT NameSysID FROM INSERTED)
    AND Fname IS NOT NULL AND Lname IS NOT NULL
    AND Fname <> ' ' AND Lname <> ' '
ELSE
    UPDATE Names SET FullName = Lname + ', ' + Fname
    WHERE Names.NameSysID =
        ANY(SELECT NameSysID FROM INSERTED)
    AND Fname IS NOT NULL AND Lname IS NOT NULL
    AND Fname <> ' ' AND Lname <> ' '

```

The same trigger in Sybase is:

```

CREATE TRIGGER Names_FullName BEFORE INSERT UPDATE
    Order 1 ON Names
    REFERENCING OLD AS OLDSTUFF
    NEW AS NEWSTUFF;
BEGIN
DECLARE pNameFormat VARCHAR(10);
SELECT NameFormat INTO pNameFormat FROM SETUP WHERE SetupSysID = 1;
IF OldStuff.Fname <> NewStuff.Fname OR OldStuff.Lname <> NewStuff.Lname
    IF pNameFormat = 'FirstLast' THEN
        SET NewStuff.FullName = NewStuff.Fname || ' ' || NewStuff.Lname;
    ELSE
        SET NewStuff.FullName = NewStuff.Lname || ' ' || NewStuff.Fname;
    END IF;
END IF;
END

```

Personally I think the Sybase code is cleaner and more readable than the MS-SQL version. Also with Sybase you have the option in the trigger to allow a WHEN condition which will eliminate two lines of code within the trigger and will also stop the execution if the WHEN condition is not met.

```

CREATE TRIGGER Names_FullName BEFORE INSERT, UPDATE
    ORDER 1 ON NAMES
    REFERENCING OLD AS oldStuff NEW AS newStuff
    FOR EACH ROW
    WHEN(OldStuff.Fname <> NewStuff.Fname
        OR OldStuff.Lname <> NewStuff.Lname)
    BEGIN
    DECLARE pNameFormat VARCHAR(10);
    SELECT NameFormat INTO pNameFormat
        FROM SETUP WHERE setupsysid = 1;
        IF pNameFormat = 'FirstLast' THEN
            SET NewStuff.FullName = NewStuff.Fname || ' ' || NewStuff.Lname;
        ELSE
            SET NewStuff.FullName = NewStuff.Lname || ' ' || NewStuff.Fname;
        END IF;
    END

```

All I did was move one line of code from the IF CLAUSE to the WHEN condition of the Trigger.

All I can promise you at this time is the more I work with MS-SQL 7.0 the better it feels even with the changes, so there will be more to come...

Austin Drum: I have been using SQL Anywhere 5.5. I now see that version 6 has been released. Also, I hear a lot of talk about MS SQL. What are your thoughts and experiences and would you recommend one above the other?

Cowboy: The newer versions of Sybase (6.0 and better) use symmetrical multiprocessing now. This means that each query or stored procedure will use all of the processors in your server. For the most part the individual would not necessarily know the difference, but in a larger environment you can see a significant increase in speed.

MS-SQL has definitely come a long way since 6.5, but I still think the language needs to mature a bit more. The documentation of methods and examples is not very instructive. I recommend you read The SQL Server 7 Developers Guide (ISBN 0-07-882548-2).

Both databases are quite capable of doing an excellent job , both for the Enterprise and personal use. MS-SQL is more expensive per seat and from a Clarion standpoint it's a bit harder to grasp the nuances. On the other hand the Sybase language is more like Clarion and easier to understand.

[Click here](#) to submit your SQL question to Andy.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).



# Clarion magazine

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

## Feature Article

### Reporting With Clarion For Windows

by Larry Teames

Back in the old days when men were men and computers were massive mainframes requiring entire floors of buildings to house all the different pieces of hardware that were connected to them, there were also huge, fast printers.

In many cases, these printers cranked out a thousand lines (about 20 pages) per minute or more, for hours on end. Back then, you couldn't just pop up a window with a page image on it, and navigate to the specific page you wanted to see, so we printed and printed and printed because that was the only way we had of getting the information we needed out of the computer. And almost without exception, what goes into a computer eventually needs to come out in hardcopy form.

These days, a PC with perhaps 10 to 20 times the power of those old computers takes up about 2 square feet of desktop or floor space, and probably has a printer attached that prints cleaner and crisper than any printer the "old guys" even dreamed of. Most printers for PCs are not as fast as those old line printers, but they seldom need to be, since we can usually see information of interest displayed on the PC monitor if we choose.

But one thing about printing reports hasn't changed:

If it's worth putting into a computer, someone, at some time, will need to get it out of the computer, and in a format that conveys the data in the most pleasing and informative way.

In other words, although we've probably diminished the need to actually produce a hardcopy of our reports in many cases (today we can view them on our monitors instead), we still need to see that information in a well designed format.

And that brings me to the reason for this column: How to get the most out of reporting in Clarion for Windows.

#### First Things First

Reporting in most Windows based programming languages is generally handled by a report writer add-on that is used to both design and run reports. Clarion for Windows has the Clarion Report Writer add-on. And it's a fine product capable of creating some nice reports.

But, CW also has a very powerful built-in Report Formatter for designing your report structures, and an internal Report Engine which generates reports from within your CW applications at runtime. Because the report is being generated from within your application, you have considerably more control over how it operates, even to the extent of making changes at runtime.

The CW Report Engine interprets the report structure, at runtime, and automatically prints much of the report for you.



**Creative  
Reporting  
Tools**  
Serious Reporting  
Made Easy!



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



Great Opportunity!  
Salary to \$125,000+  
Pre-IPD Stock Opts  
Sunny Boca Raton

The good news is that it handles things like page overflow, page numbering, and control breaks automatically.

The bad news is that it handles things like page overflow, page numbering, and control breaks automatically. (More on this dichotomy later in this series)

## Strike Up The Band

The Clarion for Windows Report Engine prints reports using "bands". A band is a collection of controls that print together at the same time, either automatically, or when asked to do so from code you have written.

There are 7 band types available for use in creating Clarion Reports:

Page Header Band

Break Group Band

Group Header Band

Detail Band

Group Footer Band

Page Footer Band

Page Form Band

All bands except Detail Bands are typically printed automatically by the Report Engine.

### Page Header Band

Prints automatically once per page before any other bands print on the page (although the Page Form is placed on the page prior to the page header printing). There can be only one Page Header Band defined for each Report. Prints in the Page Header Area.

### Break Group Band

Does not contain controls that actually print, but rather defines the variable or file field that is used to determine when a control break occurs. There can be multiple Break Group Bands defined on a Report. To specify multiple levels of control breaks, you populate multiple Break Group bands, with the highest level of break first, followed by the next lower level of break, and so on, through the last break level you need.

### Group Header Band

Prints automatically before any Detail bands for the same Break Group print. Only one Group Header Band may be defined for each Break Group defined. Prints in the Detail Area.

### Detail Band

There may be as many Detail Bands defined in a Report as needed to print the various records and/or sets of information you require for your report. Prints in the Detail Area.

### Group Footer Band

Prints automatically after the last Detail band for the same Break Group prints. Only one Group Footer Band may be defined for each Break Group defined. Prints in the Detail Area.

### Page Footer Band

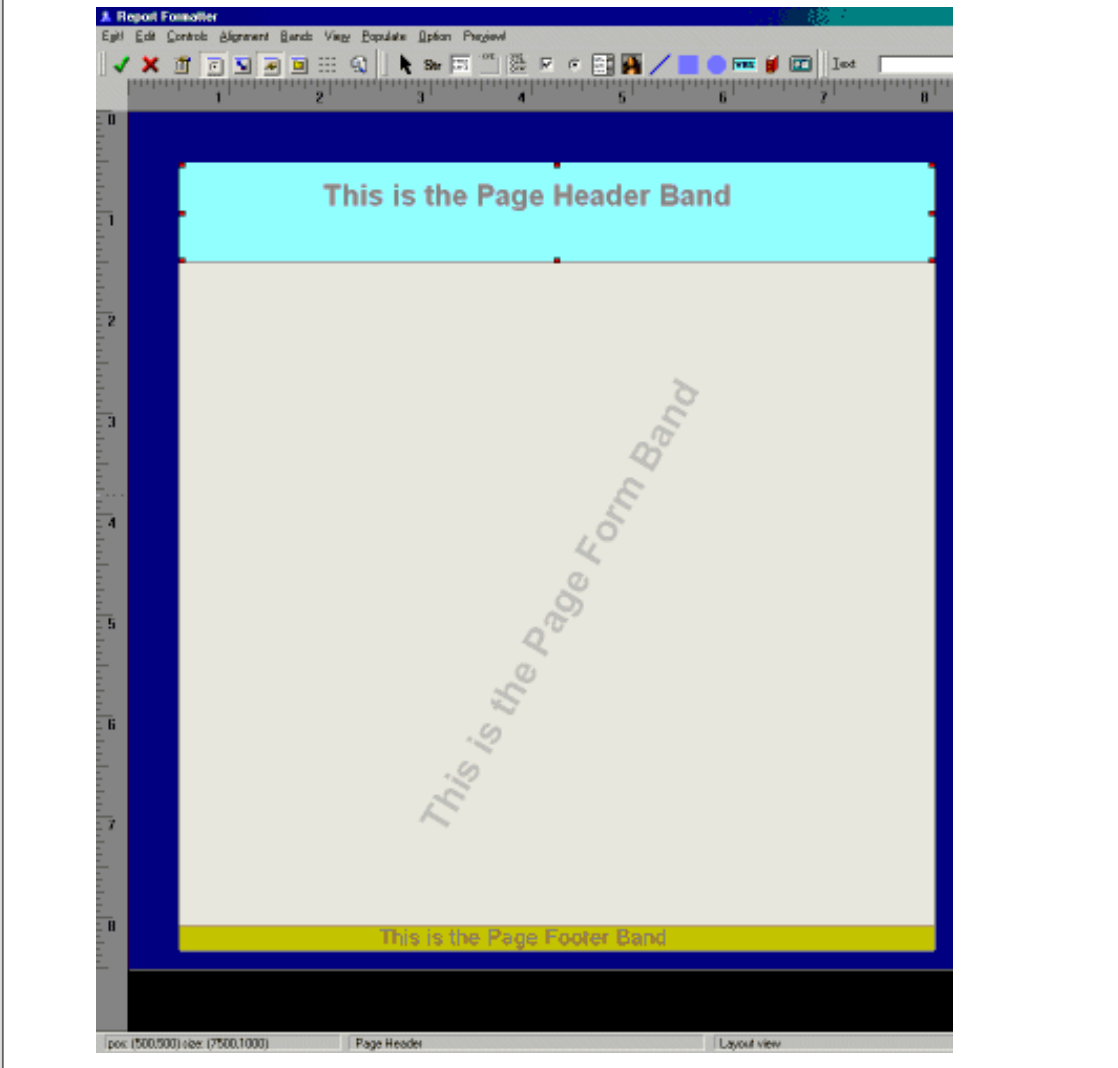
Prints automatically on each page. Generally, this band is generated immediately after printing the last Detail or Group related band that will fit in the current page's Detail area (more on exceptions to this later). Prints in the Page Footer Area.

### Page Form Band



This band is generated only once each time the report is run. This occurs automatically and immediately before the first band of any other type is printed. Although this band is generally not needed or used in most reports, it can be used to produce "form" layouts where lots of lines and boxes are used to represent a preprinted form, and/or for printing watermark images on each page of the report. The Page Form Band Prints in the Page Form Area of each page of the report.

Figure 1. Report Formatter band view (click on image for a full-size version).



## The AREAs Of A Report

As noted above, each of the 7 band types print in only one Area of a report.

The 4 Areas of a Report structure are:

1. Page Header Area
2. Detail Area
3. Page Footer Area
4. Page Form Area

Areas are a way of defining where on the report each band is limited to printing.

### Page Header Area

This is where the Page Header Band prints. This area can be located anywhere on the printed page, and can extend from top of page through bottom of page, although it is typically (by default) placed above the

**Detail Area.** This area actually defines the width, height, and location of the Page Header band on the report.

#### Detail Area

This is where all bands except the Page Header, Page Footer, and Page Form print. In other words, all Detail bands, Group Headers, and Group Footers print in this area. As each of these band types print, they fill the Detail Area from top to bottom. When there is no longer enough space remaining in the Detail Area to accommodate the next Detail or Break band to be printed, page overflow occurs, causing the Page Footer to print, followed by an eject to the next page, and the printing of the Page Form and Page Header bands.

#### Page Footer Area

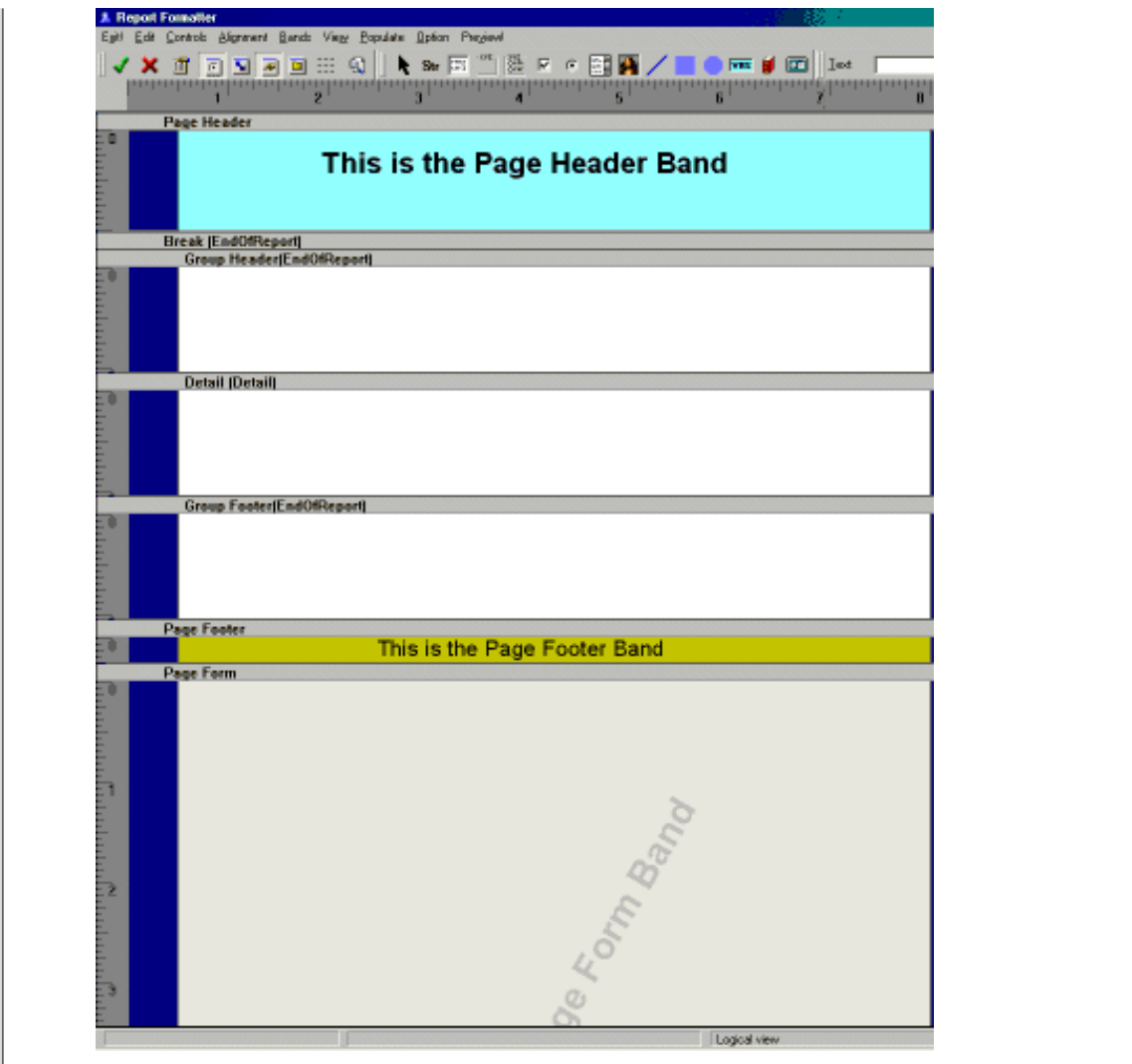
This is where the Page Footer Band prints. This area can be located anywhere on the printed page, and can extend from top of page through bottom of page, although it is typically (by default) placed below the Detail Area. This area actually defines the width, height, and location of the Page Footer band on the report.

#### Page Form Area

This area is generated only once per report (before any other band prints on the report), but prints on every page of the report. It typically extends from the top of page through the bottom of page. This area/band prints behind all other bands of the report and defines the width, height, and location of the Page Form band on the report.

You can use the Report Formatter's Page Layout View menu option to alter any of these Areas. When you alter the size and/or location of the Page Header Area, Page Footer Area, or Page Form Area, you are also modifying the actual size and location of the associated band for that area.

Figure 2. Report Formatter layout view (click on image for a full-size version).



However, when you modify the size and/or location of the Detail Area, you are actually changing the location and size of the main body of the report where all bands print except the Page Header, Page Footer, and Page Form bands print. This area corresponds to the Formatter's menu option Edit|Report Properties|Position Tab.

In the examples above, I have set the background colors of the various areas to make them more identifiable. The CYAN area is the Page Header area, the GREEN area is the Page Footer area. I have left the Detail Area with no background color so that you can see the Page Form area, which I have colored GREY.

Now take a quick look at the source for the Report structure for the example report layout so far. You can do this by pressing the "..." button to the right of the REPORT button on the Procedure Properties dialog.

Going directly to the source code for the report structure allows you to see exactly what the formatter has generated, and also allows you to make manual modifications, if desired.

Listing 1. The report structure.

```
Report REPORT,AT(500,1500,7500,6563),PAPER(PAPER:USER,8500,8500), |
  PRE(RPT),FONT('Arial',10,,FONT:regular,CHARSET:ANSI), |
  THOUS
  HEADER,AT(500,500,7500,1000),COLOR(0FFFF80H)
  STRING('This is the Page Header Band '),AT(1427,167), |
  USE(?String1),TRN,CENTER,FONT(,20,,FONT:bold,CHARSET:ANSI), |
  #ORIG(?String1)
  END
EndOfReportBreak BREAK(EndOfReport)
  HEADER,AT(0,0)
  END
Detail  DETAIL
  END
  FOOTER,AT(0,0)
  END
  END
  FOOTER,AT(500,8073,7500,250),COLOR(0FF80H)
  STRING('This is the Page Footer Band'),AT(1990,-10), |
  USE(?String2),TRN,CENTER,FONT(,16,,CHARSET:ANSI), |
  #ORIG(?String2)
  END
  FORM,AT(500,1490,7500,6583),COLOR(0EAEAEAH)
  STRING('This is the Page Form Band'),AT(313,21,6573,6563), |
  USE(?String3),TRN,FONT(,28,COLOR:Silver,FONT:bold,CHARSET:ANSI), |
  ANGLE(600),#ORIG(?String3)
  END
END
```

I use this approach quite often to create things like duplicate bands or groups of controls, to duplicate the X/Y positions and/or size of some controls (using copy and paste) to other controls, and other modifications that are simply not possible or too cumbersome from the Report Formatter window.

For example, have you ever placed a BOX control on a report only to have it appear as an incredibly small spec that defies selection with the mouse cursor? Simply go to the source for the structure and change the width and height of the control to a value that makes the control large enough to get a mouse cursor on. What could be easier?

Hopefully, as this series of articles progresses, you'll begin to feel more "in control" while creating reports, and also learn some neat tricks that will make your report designing efforts less frustrating.

Next month: Exploring the Report Formatter

---

Larry Teames is an independent software developer, and one of the four founding members of Team TopSpeed. He is also president of [Creative PC Solutions, Inc.](#) which markets the popular Clarion 3rd party product Creative Reporting Tools.

Copyright © 1999 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than [www.clarionmag.com](http://www.clarionmag.com), email [covecomm@mbnet.mb.ca](mailto:covecomm@mbnet.mb.ca).