



Clarion magazine

Volume 1, Number 6 - July 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

[A Plan For Eliminating Bugs](#)

In his first Clarion Magazine article, Bruce Gilham weighs in with a checklist of bug-avoidance techniques.

Posted on July 6, 1999

[DAB - File Manager III](#)

David Bayliss concludes his three part series on the ABC FileManager with the meat of the class: the dictionary interaction and the file access itself.

Posted on July 6, 1999

[Clarion Advisor - Drive Free Space](#)

If you've ever needed to determine how much free space there is on a drive, and you're doing 32 bit development, you'll want to get this freeware drive freespace dll from [TS Resources Inc.](#)

Posted on July 6, 1999

[The SQL Answer Cowboy](#)

The SQL Answer Cowboy answers questions about the AS-400 and capitalization triggers in MS SQL, and compares some of the popular SQL databases.

Posted on July 13, 1999

[Larry Teames On Reports](#)

In this installment Larry Teames looks at summary reporting and creates a report that can be printed in two different formats.

Posted on July 13, 1999

[Customizing Clarion5's Editor And Menus](#)

John Morter shows how to customize your Clarion development environment with editor keystrokes and a menu of your favourite utilities.

Posted on July 13, 1999

[Photo Gallery](#)

First in with a collective mug shot - the Raleigh Clarion Users Group. Way to go, guys!

Posted on July 13, 1999

[Publishing Schedule](#)

As you've probably noticed Clarion Magazine has usually, but not always, been published on Mondays, four times per month. The publishing frequency will remain the same, but the magazine will now come out on Tuesdays.

Posted on July 13, 1999

[Product Review: SearchFlash](#)

SearchFlash is a set of templates, procedures, and data files that make it relatively easy to add useful searching and tagging to any Clarion browse.

Posted on July 20, 1999

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

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

[The Other Way To Use OLE](#)

Can't wait for the new Clarion OLE layer? Jim Kane shows how to work directly with OLE using the Windows API.

Posted on July 20, 1999

[The Clarion Advisor: Debug Redux](#)

You may think you know all the tricky ways to coax debugging information out of your apps. But there's more...

Posted on July 20, 1999

[Working With Control Files I](#)

Steve Parker explains how ABC control file handling differs from legacy code, and lays the groundwork for Nik's upcoming article.

Posted on July 27, 1999

[Clarion Challenge String Parser **Final** Results](#)

At last, the results of the Clarion Parser Challenge! Five entries made it through to the end, and the winner is...well, you'll just have to follow the link.

Posted on July 27, 1999

[Clarion News, July 1999](#)

News, notes, and product announcements from the Clarion world.

Posted on July 27, 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, email 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

The Art of Software Development: Eliminating Bugs

by Bruce Gilham

This is part one of a series covering various rules that the reader may find useful or profitable.

There is a great satisfaction in seeing one's ideas put to use, and there is equally great disappointment when a design never sees the light of day. The art of software development goes way beyond writing good code. It means writing the right code, at the right time, for the right user, with the right documentation.

There are a million ways to mess up the development process. No set of rules is perfect. I wrote my first "custom" application while a hardware tech at Memorex. Since then I have written dozens of applications, many of which are still in operation. My largest is a production control system that tracks 200 million dollars per year in PC board production. My smallest might be FastWrap, a program that summarizes commercial production expenses for advertisers such as Toyota and Pepsi.

A few years ago I started a list of what amounts to all the mistakes I made as a software developer. I say this to avoid any accusations that I might consider myself to be an authority on "how to write computer programs." If anything I am an authority on how not to develop computer systems. Every one of the rules in this and the following articles was discovered in the school of hard knocks. Each one is on the list because violating it got me in trouble.

Rule #1. If the user says it's a bug, it's a bug! This is obvious when selling an off-the-shelf product, but not so obvious for in-house or custom projects. This rule becomes impossible only when there are warring factions within the company.

This almost always comes in the form of the software doing something unexpected. One could say that all undocumented features are bugs. This is perhaps an exaggeration. Or perhaps not.

When the software does something unexpected, it upsets the user. Always! It also makes enemies for the MIS department or consultant. What users want from a computer is predictability. They want it to do the "same thing" every time. By "same" they really mean "What I expect."

Users are quite happy to work around known bugs, if they understand the bug and decide that they would rather work around it than deal with the cost and/or trouble of fixing it. For instance, if they know that parts of a report are OK then they may be willing to overlook obvious flaws.

I discovered this rule when I replaced a consultant who had been fired simply because he argued about what was and was not a bug. This was a ridiculous situation since the consultant was paid by the hour; why didn't he just fix the bug?

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

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

In fact, he should have been delighted! Knowing what is a bug is half the problem because...

Rule #2: The harder a bug is to detect, the more the users will object to the system. You really only have to let the users down a few times and they will distrust the system forever. This makes the introduction period critical. Users should be clearly warned when the system is unstable. There is actually a scale of system quality: a) behaves exactly as expected b) usually behaves as expected, c) does consistently weird things, d) does unexpected weird things.

Sometimes developers have to guess at what the user really wants or how the compiler will really translate the source code. It is a good idea to "surface" such issues and make sure that the suspect data is clearly visible to the user.

One of the smartest systems I wrote was for a debt collection law firm. The program calculated interest on default accounts. The rates and even the rules changed depending on various actions in the case. The idea was to calculate the account balance automatically. And correctly. It wasn't until I surfaced every calculation on a single screen that we were able to get a perfect answer. As long as I hid the actual calculations, there were endless revisions. This single screen saved the client thousands of dollars in reprogramming fees. It was so impressive that they used it to close their clients.

Therefore, you have the corollary...

Rule #2a: Anything that you are unsure about should be visible to the users.

Of course, we developers want to look good. Of course, we want to appear omniscient. The truth is that we can't be expected to read user's minds; but we can do a pretty good imitation thereof. The trick is to surface everything that is uncertain.

In my example above, no one had the total answer. It was only when we saw wrong results that we knew to look for bugs. Once the calculations were surfaced, it was short work to fine tune the program. We manually redid the calculations in chronological sequence until the difference appeared. Then we had the bugged calculation.

What was interesting was that the users appreciated having the calculations to hand and actually preferred to have the extra data in front of them. By surfacing what I, the programmer, didn't understand, the users gained comprehension about something that they didn't fully understand either.

Besides the areas of uncertainties that come from users, there are the areas of platform unknowns. Who has grasped every nuance of the Clarion compiler? Never mind a really complex and disorganized platform like Windows. Modern programs are too large and complex for any single dweeb-superprogrammer to grasp in its entirety, much less for us overworked applications programmers.

Top Speed and other manufacturers do their best to document their products, but the average programmer will inevitably have gaps, either from poor documentation or simply from not reading the documentation to hand.

So, surfacing uncertain data is beneficial because...

Rule #3: The key to successful implementation is maintainability and testing. This has two immediate corollaries...

Rule #3a: No programmer can test his own code. This corollary should be obvious, but is usually overlooked. The quality of a system comes from the testing, not from the programming. Maintainability comes from programming skill.

With adequate testing, any programmer's work can be made usable. The test of the programmer comes when someone else has to alter his code. Or even when he has to alter his own code. It is a brutal statement to

note that many times old programs are simply discarded as unmaintainable.

Testing should be done by people who will use the system. The more testing, the better the system, period. Nothing, absolutely nothing, guarantees a usable system like testing. With this, of course, are reports of what was found.

Rule #3b: Test for success. The most common failure by testers is to a) test to the first error and quit and b) report only failures. The first is simply a waste of time and is an indication that the user really doesn't want the proposed system. Even if there are 20 things to test in a program and the tester only gets access to three of them, the project will advance much faster if he reports on the three that work.

If an area works according to spec, it should be so reported. Failure to do so invites changes. There are two reasons for implementation bugs: either the programmer misunderstood the specifications or the programmer misunderstood how the platform interprets his instruction (assuming that the system was not sabotaged, or the programmer interrupted in the middle of a critical bit of code).

I have seen programmers waste precious time fiddling with perfectly acceptable parts of a system because no one told them that it was acceptable. And I've seen programmers simply decide to change something, a disease from which none of us is totally immune.

It is especially important for a tester to note features that work better than anticipated as these are probably fortuitous errors and might be eliminated in the next version. In this case, the specifications need to be changed to match the reality of the working program.

When I started writing business applications 15 years ago, my intention was to use computers to make organizations more sane. The way to accomplish that is to make the computers more comprehensible and take out the mystery. Computers are tremendously powerful administrative tools. Misused they can drive a work force into apathy and confusion. Well understood, and with a minimum of care in the design and implementation of the software, they can easily be credited with contributing to phenomenal profits. Ironically, many computer users want the computer to "think" for them. I am reminded of the joke that recently came over the Internet.

IBM's founder Tom Watson had to be dragged into the computer age, but once committed to the electronic thinking machine he went all the way. He dictated that the company motto be: THINK. Apparently, a manager put up a sign over the sink in the bathroom: THINK. In short order a second sign appeared: THOAP.

Thank goodness we have outgrown the idea that computers are going to do our thinking for us. Just because they are such stupid machines, it takes a lot of intelligence to program a computer.

The next article I will address analysis, design and programming. What is analysis and when is it done? How much design is enough and how is it different from noodling around? Future articles will cover truly effective data validation, how to speed up development and what is the most important thing about a computer system.

Stay tuned!

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

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, email covecomm@mbnet.mb.ca.

In This Issue

[A Plan For Eliminating Bugs](#)
Posted on July 6, 1999

[DAB - File Manager III](#)
Posted on July 6, 1999

[Clarion Advisor - Drive Free Space](#)
Posted on July 6, 1999

[Clarion News, July 1999](#)
Posted on July 6, 1999



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

FileManager III – ABC Design

by David Bayliss

Welcome to the third and final installment in the ABC FileManager Design series. In this article I'll get down to the meat of the class; the dictionary interaction and the file access itself. You may want to begin by reading the [first](#) and [second](#) installments if you haven't already done so.

Record Initialisation and Validation

Whilst initialisation and validation are logically distinct tasks they are grouped together here because in the fullness of time they will form the basis of what is really one concept: business rules. People can use business rules to mean just about anything they like. I use it to mean information about the data that is not contained explicitly within the data.

Some of these rules are already handled by the file driver. For example the DUP attribute in the Clarion language (upon a key) specifies something about the data you cannot get directly from the data (although you may be able to intuit it). The Clarion dictionary gains some of its power by specifying initialisation values and validation values within a single repository. In Clarion 2.003 this information was then scattered (at code generation time) throughout the application. Any form upon a file would contain code to initialise and then validate the record buffer. In ABC this code is contained in the (derived) FileManager object.

One of the ABC aims is to keep the initialisation and validation information in one place so that if the rule is dynamic (i.e. has to be done with an embed point) then that embed only needs to be placed once and all accesses to the data obey the new rule. This is particularly important as ABC was to have edit-in-place browses and automatic updates from drop-combos. In other words, we could no longer rely on forms being around to act as guardians of the file.

A consequence of the heavy dictionary tie to these routines and the need to pander to embed code is that a number of these methods are blank; they are placeholders for template generated code. In these instances I will describe the code that I envisage will go into the derived form of these methods (and which, purely coincidentally, the templates generate.)

```
CancelAutoInc PROCEDURE(<RelationManager RM>),VIRTUAL,BYTE,PROC
```

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!

Initialising a record with an autoincrement key results (in our implementation) in a record being stored on disk to act as a placeholder for the autoincrement value. If you cancel the operation then that record needs to be deleted. The `CancelAutoInc` is the clue to the `FileManager` that the record that was initialised is about to be thrown away. The `RelationManager` parameter is a solution to the problem often referred to as the "orphaned children problem." In a 2.003 application if you insert a record with an autoincrement key, go to the child tab, insert some child records and then cancel the form, the child records persist. Of course you can't see them until you insert a new record and find it automatically gains some children! This is really scary on forms with many children where the child tabs may not even be perused during the insert. ABC gets around this problem in that the form procedure passes the `RelationManager` in to the `CancelAutoInc` and the `CancelAutoInc` undertakes to delete any children (or refuse to cancel the autoincrement) as appropriate.

The implementation is much simpler than the description: if an autoincrement has happened then either `DELETE` or the `RI DELETE` are called. In the latter case the response is noted, as an `RI` relation of `restrict` means that the autoincrement cannot be cancelled whilst the children persist.

```
PrimeAutoIncServer PROCEDURE(BYTE HandleErrors),BYTE,PROC,PRIVATE
```

`PrimeAutoInc` and `TryPrimeAutoInc` are really just two virtual hooks on the `AutoIncServer`. Non-zero for `HandleErrors` implies the server will take all possible steps to ensure the autoincrement happens. The `TryPrimeAutoInc` only is called when `TryInsert` has been called. This is never done with the shipping templates but has been added at the request of a third party.

The routine starts by checking the guard variables. `PrimeAutoInc` can be called multiple times to allow for the cases where priming is done in the browse or in the form. It also allows for inserts to be done without pre-priming of the record. This is an instance of what I call "objects with attitude." Basically the `FileManager` knows that priming should be done once, and only once, and thus it does it at the first opportunity it is asked, or at the last minute if no-one asks it!

The algorithm for autoincrementing is essentially the one used to return a unique handle in [SaveBuffer](#), find the last element and add on one. However, as you shall see, some of the little details make things a vast amount more complicated.

The first `LOOP` is a loop to allow for multiple reruns of the bulk of the procedure in the case where an attempt to autoincrement failed. The failure is most likely to be because between the reading (from disk) of the current highest element, adding on one, and then `ADD`ing the record, another station got there first and `ADDED` the record with that number! The simplest solution to this failure is therefore to try again from the beginning, and the outer loop encodes that logic. Of course you can't keep doing that for ever because something may really be wrong. If you go to the end of the outer loop, after the `ADD(File)` you will see logic to trap the error.

If this is the third failure the error manager is invoked to see if the user wants to try again. If he does you simply try again (three times), and if he doesn't you break out with a failure. If the add was successful the method simply notes the autoincrement has been done and then returns `Level:Benign` (which is zero and means OK).

The second (or inner) loop introduces a new complexity. Since it is possible for there to be multiple autoincrement keys in one file, this code has to find incremented values for each of them. So it loops on each key, executing the body of the loop if it is autoincrement. The `SaveBuffer` is important because `PrimeAutoInc` should only change those fields which are tagged in the dictionary as autoincrement (or it won't be possible to support delaying the autoinc allocation until the insert point). The code

then splits into two branches, the relatively easy one where the key has one component, and the multi-component case.

One component is handled by fetching the component any variable and assigning to `AutoIncField`. The file is `SET` to key order and the final record (or first for a descending key) is fetched. If no record is found (i.e. the file is empty) the new `autoinc` value is one, else it is the highest value plus one. It's necessary to use `AutoIncField/AutoValue` rather than just using the underlying fields as the object will later restore the buffer (which will corrupt the field values) and then perform the autoincrement assignment.

The multi-component case uses exactly the same algorithm. The complexity is in finding the "final" record because you don't want the final record, you want the final record that matches the current record buffer in all components except the last.

`ConcatGetComponents` is a simple (and ugly) way of snapshotting the leading components of a key to later see if they have changed. The method then clears the minor-most (and thus autoincrement) key high and does a `SET(K,K)` followed by a `Previous` or `Next` as before. In the `NoError` case it has to check that the record fetched did match in the leading components; if it did then it can use the AI value fetched and add on one, otherwise it knows that no records currently match the major components and thus can use one.

Having computed the new field value (using either method) it restores the buffer contents and then assigns the new autoincrement field value into place. Once that is done that for all autoincrement keys it can try `ADDing` the new record.

If this procedure looks long and horrible it is because it is. My general rule of thumb is that any procedure more than a page long is a bug. Again you can see engineering and efficiency overcoming science. I could remove the "OneComponent" arm of the `IF`, and the only effect would be a few more string compares (no big deal), but this also changes this code:

```
CLEAR(OnlyKeyComponent)
SET(K,K)
PREVIOUS(K)
```

into this:

```
SET(K)
PREVIOUS(K)
```

When you consider that the one component case is the standard third normal form case it was considered that the code verbosity was worth it. That said, the file drivers now spot the above optimisation in most instances so we may be able to simplify this code soon.

```
PrimeFields PROCEDURE,PROC,VIRTUAL
```

The default implementation of this method is blank; in the derived form the templates insert an assignment for each field that has a non-blank initialisation value in the dictionary. The `PrimeFields` routine may not assume that autoincrement has been done. Any required blanking will have been performed.

```
PrimeRecord PROCEDURE(BYTE SuppressClear = 0),BYTE,PROC,VIRTUAL
```


This method is called to prime the record whatever that means. In the current implementation that involves calling `PrimeFields` to prime the field values and then forcing the autoincrement to prime. Note that this method overrides the attitude built into `PrimeAutoInc`; when `PrimeRecord` is called a new autoincrement record will be made. The method has the facility to clear out any fields it doesn't explicitly prime or to leave them alone. This functionality is required to allow the `ViewManager` to place extra priming information into the record buffer (such as range-limited components of keys) and is controlled by the `SuppressClear` flag.

The primary case (where `AliasedFile` `&= NULL`) is quite straightforward. The interest comes when the file is an alias. Here you don't want to call the priming functions of the alias (because they don't have the required embed code); you want to call them in the "real" file. The code for this has changed in C5EEA; I am describing the new code.

First the "real" file has to be opened on this thread (it may not be), and then the file contents/position have to be snapshotted so that eventually the file can be restored. Now in the case where the clear is to be suppressed the code has to assume that the record (of the alias) contains interesting information that may be required by the field priming or autoincrement. So it has to get the information from the alias into the real file. This is done by the devious device of snap-shotting the alias file buffer and the restoring from the alias `FileManager` into the "real" file buffer. Having done this it can perform the `PrimeRecord` on the "real" file; if this is successful it needs to copy the result back to the alias file, done using the save/restore trick again. Finally the "real" file is restored to normality and closed. This may look odd: why restore a file then close it? Simply because `Open/Close` only increment/decrement counters. The `Open` only opens if this is the first open, similarly the `Close` only closes if this is the last close.

```
ValidateField PROCEDURE(UNSIGNED Id),BYTE,PROC,VIRTUAL
```

This function returns `Level:Benign` if the field is ok, otherwise it returns an error level. The default implementation only handles the alias case; the actual field validation is handled in a derived method. The `Id` is the number that would come back from a [WHERE](#) statement. The template code simply generates a CASE statement on the field number, it would be possible to produce a more sophisticated version using [WHAT](#). We went for simplicity as this is a very common place to put embed code and also `ValidateField` is hit quite frequently for control by control field validation and therefore performance was an issue.

```
ValidateFields PROCEDURE(UNSIGNED Low,UNSIGNED High,  
    <*UNSIGNED Failed>),BYTE,PROTECTED,PROC,VIRTUAL
```

This method is simply an encapsulated way of calling a range of `ValidateField` calls. It simply spools over the field numbers contained within the (inclusive) range. If one fails then the failure number is assigned to `Failed`. Again the alias case is handled by re-vectoring through the "real" file. It should perhaps be noted that for efficiency the alias code does a `SaveBuffer`, not `SaveFile`. This implicitly assumes that the field validation code will not mess with the current file state (i.e. no file I/O will be done on the primary file).

```
ValidateRecord PROCEDURE(<*UNSIGNED Failed>),BYTE,VIRTUAL
```

Another syntactic short-hand, this simply calls `ValidateFields` to ensure that every field in the record is validated.

File Driver Replacements

These methods are direct replacements for the equivalents in the file driver. They are generally there to perform advanced error handling or to ensure that other `FileManager` routines are called at the appropriate moment.

```
BindFields PROCEDURE,VIRTUAL
```

This method is called at a suitable point to bind the fields. By default the code simply performs a bind on the record buffer. The templates further override this to perform binds on any memos that are available. The call can also be overridden by the user (in C5) to bind logical names (as opposed to labels) as required.

```
Close PROCEDURE,BYTE,PROC,VIRTUAL
```

The close mechanism (tied to the open mechanism) is designed to avoid the needless opening and closing of files upon a thread. The `FileManager` therefore maintains a count of the number of times a file has been opened and closed. Upon a close it therefore decrements the counter. If this close has closed the final remaining open then the close is actually performed upon the file. The `Used` flag denotes if a file was really forced open (by an implicit or explicit `UseFile`) as opposed to just logically opened. Errors are not trapped by this routine as any real problem with a close will be picked up again when the file comes to be re-opened. For this reason ABC also eschews the `TryClose`.

```
Fetch PROCEDURE(KEY K),BYTE,PROC
```

The `Fetch` routine is really just a wrapper for a file `GET`. The error case results in the buffer being cleared. Most of the work is done by re-vectoring through `TryFetch`, and though this doesn't save a great deal of code and is marginally less efficient than inline coding it does result in greater code integrity. Put another way, the code for fetching is in only one place so only has to be fixed in one place.

```
InsertServer PROCEDURE(BYTE HandleError),BYTE,PRIVATE
```

`Insert` and `TryInsert` are just interface maps of this procedure. `InsertServer` is a fairly good illustration of the difference between the `FileManager` and the file driver equivalents. It is only really trying to do an add; all of the other code is there either to handle errors or to ensure other ABC methods get called as appropriate.

First `UseFile` is called. This registers that not only is this file logically open, it needs to be actually open. Then comes a call to `ValidateRecord`. If the record is not valid then the method returns. Note throughout ABC the fact that `Level:Benign` is zero is assumed to aid readability and brevity. There are then three cases:

1. No autoincrement keys. In this instance the record will not already exist so a new one can be added
2. There are autoincrement keys and the autoincrement has been pre-primed. In this case the record does exist resulting in a `PUT` rather than `ADD`.
3. There are autoincrement keys but they have not been pre-primed. Call the autoincrement logic to `ADD` the record (remember `PrimeAutoIncrement` does not corrupt the record buffer other than the autoinc components themselves).

There are then three error conditions to worry about:

1. **NoError.** In this case simply note that any previously primed autoincrementing has now been used and return. (Note this code assumes that `PrimeAutoIncrement` will not have left a value in `ErrorCode` if it was successful; I suspect that technically this is a bug.)
2. **DupKey.** This is the only error the method attempts to recover from gracefully, stepping through the keys and alerting the end user of any duplications that this record causes.

Everything else. Post a general (cryptic) error message to the user and return.

```
NextServer PROCEDURE(BYTE HandleError,BYTE Prev),BYTE,PRIVATE
```

Again `Next` and `TryNext` are just interfaces to this method. Since `C5EEA` `Previous` and `TryPrevious` have also become interfaces to this routine (the beauty of this method being private!). The `NextServer` and `PreviousServer` methods of earlier versions differed only in one line which has now been parameterised with the `Prev` byte.

There are only two real points of interest. Firstly the `BadRecErr` sets the EOF flag (see [GetEOF](#)). Secondly ABC has a facility whereby a held record error can be treated simply as a `Skip` rather than as an EOF (which it was in 2.003). You can argue back and forwards for hours as to whether it is better to display a browse with information missing or to abort the display. ABC takes the approach that that decision is best left in the hands of the developer (as the real answer is probably dataset specific) and so provides a property for her to register her decision.

```
OpenServer PROCEDURE(BYTE HandleError,BYTE IncrementUsage=True,
                    BYTE ForceOpen=False),BYTE,PROC,PRIVATE
```

`Open` and `TryOpen` are interfaces to this method. The second and third parameters are really for `UseFile`. They allow an actual open to be forced without a logical open happening. Specifically, setting `increment usage` to false means that a corresponding close is not required. `ForceOpen` is used to force the file open even when the `LazyOpen` status would suggest the open can be deferred. Note that `ForceOpen` is quite safe as the routine takes a failure to open because the file is already open as a success. Again most of the work of the routine is in error handling, and in this case some moderately sophisticated recovery is allowed for.

1. **NoError** or **FileOpen**. Both treated as a success, internal state variables cleared (on this thread).
2. **RecordLimitError**. This code is just there for the evaluation edition. An attempt has been made to open the file in read/write mode with greater than the set number of records in the file. The code therefore sets the `OpenMode` to read-only and cycles (the loop will then cause another open attempt).
3. **NoAccessError**. Read-write access could not be acquired so the system tries to open the file in read-only mode (having first warned the end user)
4. **NoFile**. Provided the create mode has been set the routine will attempt to create the file, if that fails a fatal error is thrown.
5. **BadKeyErr**. For those file drivers with independent key files (notably Clarion) a corrupt key is non-fatal and the system will try to rebuild the keys so that processing can continue.

The `UNTIL 1` at the end of the loop means that any code falling through to the end of the loop will cause the loop to terminate. The `LOOP` is not a real loop; it is simply there to allow some of the recovery routines to attempt to open the file again without a `GOTO` statement. It should be noted that a `CYCLE` statement bypasses the loop tail termination

condition but not the loop top termination condition.

```
Position PROCEDURE( ),STRING
```

This method differs from the file driver equivalent in that it will issue a `UseFile` and if a primary key is available it will use that to form the position, or if there is no primary key it will use the file itself. In general (and increasingly) the ABC system assumes (and functions most efficiently providing) that all files in the system have a primary key. Note that this position string can only be used to perform a [TryReget](#); it is not as general as the Clarion language `Position`. Whilst this functionality restriction does not give us much presently it will eventually allow extra efficiencies within the forthcoming `FileClass`.

```
TryFetch PROCEDURE(KEY K),BYTE,PROC
```

`TryFetch` only really does a `UseFile` before passing control to the file system, although since C5EEA it has also performed a `SetKey` in debug mode purely to verify that the key passed in is valid for this file. (The ? is one of my favourite C5 features; it allows you to write code very cleanly which will not be executed when debug is turned off. This allows you to put in quite a few safety checks with zero overhead in the final shipping code.)

```
TryReget PROCEDURE(STRING Position),BYTE,PROC
```

Perform a `Reget` from a string provided by `Position`. The `Reget/Position` pair give the `FileManager` user one extra piece of encapsulation: independence from key structure. Without them you need to know from outside the class how to uniquely identify a record. This illustrates an important aim: localising information to reduce maintenance.

```
UpdateServer PROCEDURE(BYTE HandleError),BYTE,PROC,PRIVATE
```

`Update` and `TryUpdate` are interfaces to this procedure. Clearly this is similar to [InsertServer](#). The main extra comes from concurrency issues. ABC implements a technique called optimistic concurrency. Put simply, this means the algorithm assumes that no one else will ever change the record being edited locally, and then panics if they did. It relies upon a `WATCH` having been issued before the record (now being updated) was fetched. In standard ABC usage the `WATCH` is issued before the view `REGET` in the browse `UpdateViewRecord` method.

To handle this the code first takes the position of the current record, then it tries the `PUT`, if this returns a `RecordChangedErr` then the user is notified. (To help the end-user the form template passes in 2 as the `HandleError` value which prompts the use of a fairly verbose error message which tells the user of such things as the history key). Then the record saved by the other station is loaded (i.e. corrupting the local file buffer) and control is handed back.

```
UseFile PROCEDURE( ),BYTE,PROC
```

The `UseFile` method is there simply to perform a real open (using [OpenServer](#)) if lazy open is currently on and the file is not open. This routine has one of the few bits of defensive coding in the whole of ABC; it actually preserves the file buffers across the Open call just in case the file driver (which could be supplied by a third party) corrupts the file buffer upon the open.

Conclusion

I hope these articles on the `FileManager` have helped you understand some of what we were aiming for (and have achieved) when we coded this class. It is one of our largest and most complex, and it also presently forms the base of what I call the spine of ABC: `FileManager` → `RelationManager` → `ViewManager` → `BrowseClass`. `FileManager` is also the class the developer most frequently needs to interact with (at least as much as `BrowseClass` and `WindowManager`). As such I believe an understanding of the principles involved will send you well on your way towards mastery 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.

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, email covecomm@mbnet.mb.ca.

In This Issue

[A Plan For Eliminating Bugs](#)

Posted on July 6, 1999

[DAB - FileManager III](#)

Posted on July 6, 1999

[Clarion Advisor - Drive Free Space](#)

Posted on July 6, 1999

[Clarion News, July 1999](#)

Posted on July 6, 1999



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 Advisor

32 Bit Drive Freespace Function

Courtesy Of [TS Resources Inc.](#)

If you've ever needed to determine how much free space there is on a drive, and you're doing 32 bit development, you'll want to get the [freeware drive freespace dll](#) from [TS Resources Inc.](#) (Rob Cohan).

This is a C language DLL which can be called by Clarion 32 bit applications, and which returns accurate disk free space figures for all 32 bit Microsoft Windows operating systems. It uses internal dynamic linkage to avoid the problems associated with statically linking the accurate, advanced functions which are not available in the original releases of Windows 95.

To use this DLL in your application choose Application|Insert Module from the main menu, and select a module of type ExternalLib (External Library Module). Fill in the Name and Map Include File fields as shown in Figure 1. The spc.mif file contains the prototype for the spc function.

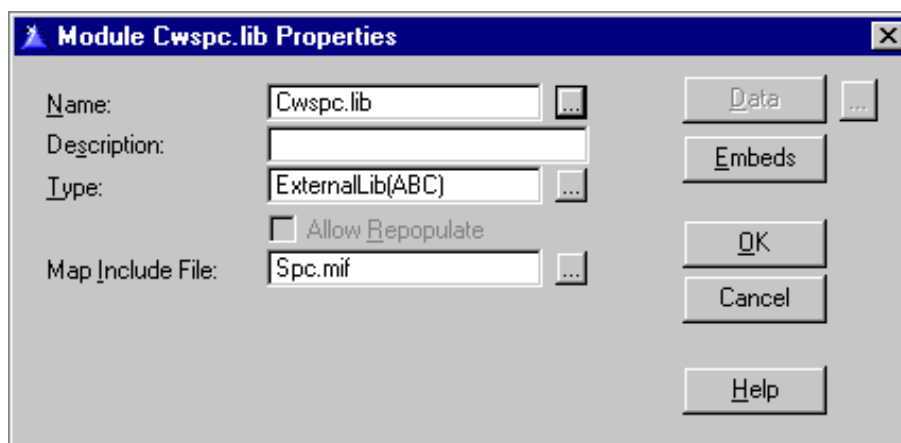
BKO
Enterprises, Inc.

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 CWSPC module properties.



The [zip file](#) contains an example application.

The `spc` function usage is straightforward:

```
result = spc( cstring *DriveLetter, long Mode )
```

`DriveLetter` is a `cstring` of at least two bytes length (one character plus the requisite terminating NULL character space.)

`Mode` can be one of the following:

- 1 for a free space call,
- 2 for the size of the drive if empty, or
- 3 for the presently utilized space.

Under Windows 2000, a `mode` of 1 is intended to get free space allocated to the present user.

`spc` returns a numeric `cstring` to allow for future variability in the size of number reported. Due to Clarion's automatic type conversions, you can use this result directly as a number in most cases, even when formatting the result, as shown in Listing 1.

In This Issue

[A Plan For
Eliminating Bugs](#)

Posted on July 6,
1999

[DAB - File
Manager III](#)

Posted on July 6,
1999

[Clarion Advisor -
Drive Free Space](#)

Posted on July 6,
1999

[Clarion News,
July 1999](#)

Posted on July 6,
1999

Listing 1. The example application code to determine drive space.

```
csDriveLetter = clip(DriveLetter)&'<0>' !turn into cstring)
FreeSpaceNow   = format( spc(csDriveLetter, 1), @n15)
TabulaRasa     = format( spc(csDriveLetter, 2), @n15)
Utilized       = format( spc(csDriveLetter, 3), @n15)
```

Many thanks to TS Resources Inc. for making this utility available!

[Download the dll and example application.](#)

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, email 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.

Ken Castleberry: Can AS-400 files be accessed from a Clarion 5 Professional application, with the only addition being the AS-400 drivers?

Cowboy: From everything I gather this is true, as long as you have the interface from the PC to AS400. This used to be PC-Talk and was quite slow. Now a more native form of communication is available and speed has increased dramatically.

Before jumping into any client/Server arena check the communication layer associated with the systems. This includes any non-NT to NT/Windows platform. Some will perform quite well (usually those on TCPIP), others will run badly.

Scott Jordan: How do I force a field in the table to uppercase or capitalize a word? Declaring an attribute in the data dictionary does not seem to have any effect. I am using MS SQL 7

Cowboy: After searching everywhere, I finally asked a good friend Ben Williams, since he has been saddled with MS SQL for quite a while. He confirmed my suspicions. In MS SQL 6.5 or 7.0 field attributes are unavailable, and the only method to force upper case or capitalization is via a trigger or your program. This is a major shortcoming in MS SQL in my opinion.

Here is a trigger that will force uppercase on a Name/Address/City/State:

```
CREATE TRIGGER UppercaseAddress
ON Names
FOR INSERT, UPDATE
AS
If Update(Fname) or Update(Lname) or update(Address)
or Update(State) or Update(City)
update Names
set LName = UPPER(Lname),
Fname = UPPER(Fname),
Address = Upper(Address),
City = Upper(City),
State = Upper(State)
Where Namsysid = any(select Namesysid from Inserted)
GO
```



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!

Now you should also use UPPER on all your Clarion screens...

James Fortune: I was interested to read your comparison of Sybase and MS SQL, especially as I believe that the latter is derived from Version 3 of the former. My question follows on from this. In your informed view, how do Oracle and Informix fit into the picture? How would you rank these in terms of effectiveness? Especially in terms of ease of use with CW? I'm assuming that the backend platform is NT. But what about scalability? Is it still true that MS SQL will only work on NT whereas Oracle will work on anything including Linux? What about the others?

Cowboy: Yes, MS SQL does require NT. Another reason to prefer Sybase over each of these is that Sybase is scalable and also has versions for Unix and Linux. I can speak more on Oracle rather than Informix so here is my best answer.

Oracle and Informix are more of a mainframe type technology. Both are platform independent and expensive in cost and maintenance. Oracle can be difficult to say the least; a lot of the convenience we enjoy is lost in Oracle. At the moment the reference manuals that I have had to purchase to know the changes for Oracle 8.0 are somewhere around 30lbs in four books.

One of the pros of Oracle is it's quite scalable. You can continue with Oracle throughout terabytes of data with quite excellent performance. If you are going into a Oracle shop and working with Oracle, here is a list of books that I find essential:

Oracle 8 Tuning ISBN: 1-57610-217-3

Oracle8 The Complete Reference (Oracle Press) ISBN: 0-07-8822406-0

Oracle8 DBA handbook ISBN: 0-07-882396-x

[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, email covecomm@mbnet.mb.ca.

In This Issue

[The SQL Answer Cowboy](#)

Posted on July 13, 1999

[Larry Teames On Reports](#)

Posted on July 13, 1999

[Customizing Clarion5's Editor And Menus](#)

Posted on July 13, 1999

[Photo Gallery](#)

Posted on July 13, 1999

[Publishing Schedule](#)

Posted on July 13, 1999



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

by Larry Teames

Over the next few months, I will be designing a number of reports to illustrate approaches that can be used to handle various reporting issues, both common and uncommon. To make it easier for you to follow and later review what I cover, I have created a small dictionary (BREAKS.DCT), and the TPS files to go with it (LETTERS.TPS, BREAK1.TPS, and BREAK2.TPS). Also the resulting APP file(s) that I create will be available.

This month's APP file is [CMEXO1.APP](#).

I will be using Clarion5 (ABC) to build these applications, but in most cases I will use the legacy (non-ABC) embeds when placing source code in the report procedures. Personally, I find that the legacy embeds have better descriptions and finer resolution related to what I want to do and where I want to do it. Also, using this method you can look at the source file for the procedure and locate the ABC embed that correlates to the legacy embed that you just entered. This provides a very nice approach to making the transition from using legacy embeds to using ABC embeds.

Summary Reporting

This month I've built a report that allows the user to decide whether they want to print invoices in detail form (all invoices with totals at the customer and report level), or in summary form (totals only).

I like to make a window for selection of report criteria (instead of using the report's progress window) when there are report variations available which might cause the user to run the report multiple times with different options chosen (see the `SummaryReportSelect` procedure below). I tend to have a menu option call the selection window, then the selection window calls the appropriate report (or the same report with different parameters, as in this case). This allows the user to run the report, then be returned to the selection window where they can change the selection criteria and run the report again, or exit the selection window when no more reports are needed.

The report procedure, `SummaryReport`, takes a single `BYTE` parameter (`SummaryRequest`) which is `TRUE` when the user selected `Summary Report`, and `FALSE` when they did not request the summary form of the report. Note that all the changes to the report are performed in the source code entered into the `After Opening Report` embed. I'll go over this code in detail later in this column.

Having decided that I wanted a single report to provide both detail and summary formats, I knew that I would need to `HIDE` all the information in the `Detail` band when the summary form was requested (when using the built-in breaking and totaling mechanism of the report engine, tallying of values is directly related to the printing process, so you have to continue to print the `Detail` band to get the totaling to occur). To

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!

more easily accomplish this, I placed a `GROUP` around the controls in the `Detail` band. This allows me to `HIDE` all controls in that band with a single statement, by simply hiding that `GROUP`. Note that I removed the display text of the `GROUP`, and unchecked the `Boxed` option so the box around the group doesn't print.

To set up the totaling, I used the `Surrounding Break` menu choice twice (from the `Bands` menu item): once to create the break on `Name`, and again to create the break on `EndOfReport` (I pressed `"..."` from the `Break` dialog, then `New`, to create the `EndOfReport` variable).

After creating these break groups, I selected each of these breaks and created a break footer for each. Then I added the controls I wanted to print at total time. For the amount fields, I set both to `sum total` type, to tally on printing the `Detail` band.

To facilitate the source code statements I would need, I added `USE` names to the `Detail` band (`?DetailBand`), and the `Name` break footer (`?NameFooter`).

The Tricks

This report will not only produce two different formats in terms of content, but will also modify the vertical spacing of the `Name` footer when the `Summary` format is requested (so that there is no extra whitespace between summary total lines). Additionally, the name of the report will be changed to identify the specific report format being printed.

All that's left is to enter the source code that will do all the work into the `After Opening Report` embed, as shown in Listing 1 (the numbers at the end of each line correlate with the legend that follows describing the purpose/effect of the code).

Listing 1.

```

IF SummaryRequest = True           !(1)
  SETTARGET(Report)                !(2)
  ?ReportHeading{PROP:TEXT} = 'Example Summary Report'  !(3)
  ?NameFooter{PROP:YPOS} = 0 - ?BR1:Name:2{PROP:YPOS}  !(4)
  ?NameFooter{PROP:HEIGHT} = ''    !(5)
  ?NameFooter{PROP:MINHEIGHT} = '' !(6)
  ?DetailGroup{PROP:HIDE} = True    !(7)
  ?DetailBand{PROP:HEIGHT} = 0      !(8)
  ?DetailBand{PROP:MINHEIGHT} = 0  !(9)
  SETTARGET                          !(10)
END

```

1. Only change the report characteristics if the user selected the `Summary` format (which is passed to the `Report` via a procedure parameter).
2. Set the `Report` as the target of subsequent field equates so that the compiler knows that I am referring to controls on the `Report` instead of (by default) the active window.
3. Change the text of the report name string.
4. Causes all controls in the `Name` break footer to be adjusted "up" so that they are printed relative to `YPOS` zero (by converting the band's `YPOS` to a negative value equal to the `YPOS` of one of the topmost controls on the band).
5. Sets the `HEIGHT` of the band to the default setting (as opposed to setting it to zero, which would not be what I want). This setting causes the band to be only as tall as the sum (`YPOS + HEIGHT`) of the bottommost control on the band. Here I'm assuring that there is a minimum of vertical space between the lines printed. Note that the detail format generates significant space at the bottom of this band to better define where a break has occurred.

In This Issue

6. I also need to do the same to the minimum height of the band, since it was originally set to a fixed height when the report opened.
7. Next I hide all controls on the Detail band so they don't appear on the report when this band is printed.
8. I also want to set the PROP:HEIGHT of the Detail band to zero so that it causes no vertical movement when printed.
9. I also set the minimum height to assure no surprises at runtime.
10. Finally, I set the "target" back to the default (the currently active progress window).

Now I have a single report procedure that allows the end user to print two different formats of reports.

Obviously, a report with more controls and totals would likely require more source code to accomplish this, but the principals and approaches would remain the same as what I've illustrated here.

Next month, I'll tackle another example of a common reporting issue, so don't miss it!

[Download the source code](#)

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, email covecomm@mbnet.mb.ca.

[The SQL Answer Cowboy](#)

Posted on July 13, 1999

[Larry Teames On Reports](#)

Posted on July 13, 1999

[Customizing Clarion5's Editor And Menus](#)

Posted on July 13, 1999

[Clarion News, July 1999](#)

Posted on July 13, 1999

[Photo Gallery](#)

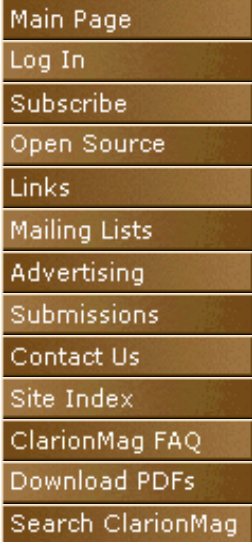
Posted on July 13, 1999

[Publishing Schedule](#)

Posted on July 13, 1999



Clarion magazine



Feature Article

Customizing Clarion5's Editor And Menus

by John Morter

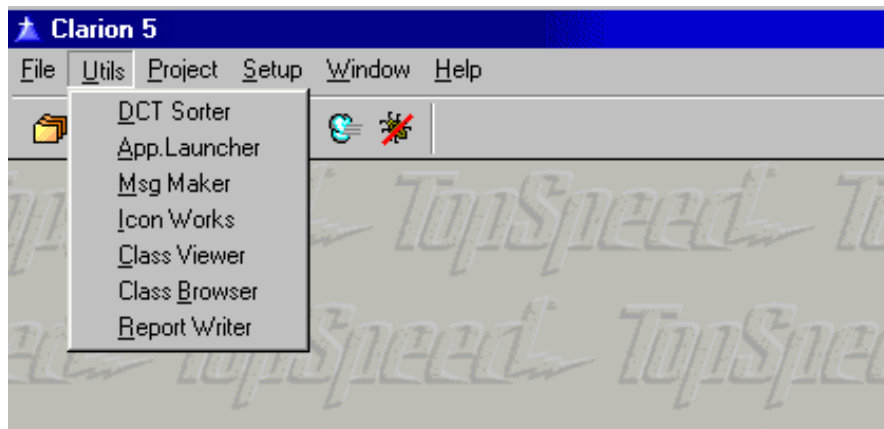
In a recent [Clarion Advisor article](#) Bruce Wells explained how to go about configuring the colour highlighting used by the Clarion 5 Editor. I'm going to take this general concept a bit further and show you other ways to personalise your working environment.

A Custom IDE Menu

A simple, but very useful feature provided by TopSpeed lets you add user defined choices to the Clarion IDE application frame menu. For example, I've added a Utilities menu to the IDE that gives me handy access to various commonly used applets. My own Utilities menu now fits in neatly between the standard File and Project menus, as you can see in Figure 1.



Figure 1. A custom Utilities menu.



Here's how you can create a menu to suit your own requirements:

Step 1. You're about to modify the CLARION5.INI file, so the strongly recommended cautious approach is to make a copy of the original file first - just in case. (I always do this by creating a duplicate of the file, replacing the last character of the file extension with an underscore. In this case, creating CLARION5.IN_)

Step 2. Open CLARION5.INI, which you'll find in your BIN folder, with the text editor of your choice and search for the section headers [User Menus] and [User Applications] where the square brackets are part of the section header construct.

The [User Menus] section provides the descriptive part of your new menu. For example, the [User Menus] section of my CLARION5.INI file is shown in Listing 1. You can readily see how this matches with the result in Figure 1.

Listing 1. The [User Menus] section of my C:\CW5\BIN\CLARION5.INI file.

```
[User Menus]
_version=41
1=&Utils/&DCT Sorter|DctAS
2=&Utils/&App.Launcher|Lnchr
3=&Utils/&Msg Maker|MakeMsg
4=&Utils/&Icon Works|IconPro
5=&Utils/&Class Viewer|CallTree
6=&Utils/Class &Browser|ClsBrwsr
7=&Utils/&Report Writer|CWRW
```

Before you go any further, note the line immediately beneath the section header. This is some sort of magic number used by TopSpeed for some sort of under-the-bonnet reason. Don't be tempted to fiddle with this number - just leave it as it is. (You'll find that the standard contents of this section in CLARION5.INI consists only of the first two lines above, and the magic number does change between Clarion versions.)

&Utils is the menu name which will be merged in with the IDE menu, using the standard Clarion convention of identifying the accelerator key character with an ampersand. The leading numbers simply differentiate the menu items within Utils.

The next parameter is the name of the menu item within the Utils menu, again using the ampersand prefix convention to identify the accelerator key.

The last parameter provides the link between this section and the [User Applications] section, which I'll look at next.

But first, go ahead and edit in your own user defined menu. You might like to have handy access to [Le Schmoo's Class Browser](#) (highly recommended) or to a Calculator or Calendar utility - or whatever!

Step 3. Now take a look at the [User Applications] section. The standard content of this section in CLARION5.INI is shown in Listing 2.

Listing 2. The default [User Applications] section of CLARION5.INI.

```
User Applications]
_version=41
CWRW=c5rw %f %a
```

The same rule applies for the magic number line - don't mess with it!

The line starting with CWRW is a hint from TopSpeed, and shows some of the more advanced features available to you in customising your own menus. The %f is an expansion macro; it will be replaced with the file and path-name currently opened by the IDE. Similarly, the %a is another expansion macro which will be replaced by the current Application or Project name. (See the C5-PG.pdf for more details of these and other expansion macros.)

As I mentioned earlier, the last parameter on lines within the [User Menus] section provides a link with the contents of the [User Applications] section. See my [User Application] section (Listing 3) to understand this better.

Listing 3. The [User Applications] section of my C:\CW5\BIN\CLARION5.INI file.

```
[User Applications]
_version=41
DctAS=C:\CW5\Utils\DctAS.exe
Lnchr=C:\CW5\Utils\Lnchr.exe
MakeMsg=C:\CW5\Utils\Make_Msg.exe
IconPro=h:\WinUtils\IWPRO\IWPro.exe
CallTree=C:\CW5\Bin\CallTree.exe
ClsBrwsr=C:\CW5\Utils\ClsBrwsr.exe
CWRW=c5rw %f %a
```

I'll use the second last line of this example to explain what's going in. (It's pretty simple, and you're probably all way ahead of me by now - but I'm gonna be pedantic!)

ClsBrwsr provides the link between the two sections. It tells the [User Menus] section where to find the executable identified in that section as Class Browser.

Notice that I've picked up on TopSpeed's hint to call the Clarion Report Writer by creating a menu item in [User Menus] which links with the CWRW line provided in the [User Applications] section.

Step 4. That's all there is to it. Now save your changes to CLARION5.INI, restart the C5 IDE and make use of your nifty new menu items.

TIP: If changes you've made to your CLARION5.INI file are not reflected in the IDE menu then you'll need to give CLARION5.exe a bit of encouragement to recognise your work. The IDE is not actually reading your CLARION5.INI file each time it starts. Instead, whenever it detects a change in various configuration files it summarises all this stuff into a file named CLARION5.DAT - and it reads that file instead. You can safely delete the CLARION5.DAT file to force its recreation, this time with your CLARION5.INI file changes included.

If the idea of deleting this file makes you a little uneasy, then take the just in case approach again and rename it to CLARION5.DA_ instead. (To be honest, that's what I always do!)

Customised Editor Commands

If, like me, you've been around for a while in the programming game then it's quite likely that, like me, you've got the commands of a favourite text-editor built into your finger-tips, if not into your brain.

For me it's an old Data General editor from my dim, dark, past COBOL programming days. For others it may be WordStar or the XTGold editor, or something else that you've used happily for years - whatever!

I'll show you now how you can influence Clarion5 with flavours from your old favourite text editor. (And, once again, it's pretty simple, once you

know how.)

Step 1. This time you'll be working with C5EDT.INI, which you'll find in your BIN folder. Again, I recommend you start out by taking a just-in-case backup copy before making any changes.

When you open C5EDT.INI with your favourite text editor you'll find something like the following, (actually, probably exactly like the following).

Listing 4. Contents of the standard C5EDT.INI file.

```
[Text Editor Configuration]
_version=43

[Key Mapping]
CharLeft=LeftKey
```

I'm sure you've got the message by now - don't mess with the magic number line!

The lines listed within the [Key Mapping] section have the general format of ...

```
Edit-command-effect = ↵
Keyboard-combination [; Keyboard-combination ...]
```

For example;

```
CharLeft=LeftKey
```

means;

"To move the cursor left one character" = "Press the Left (arrow) Key"

Step 2. Have a look at the entries in this section. Even if you don't intend to change the way the Clarion5 editor works, you'll still likely discover lots of useful keyboard commands you didn't know about, (and which aren't documented elsewhere!).

Step 3. Assuming you do want to put your own stamp on the way the Clarion5 editor works, then there are some important things you need to keep in mind:

There are certain keyboard combinations that are built right into the Clarion5 editor or are part of the standard Windows convention set. You are strongly advised not to redefine these for your own use. At best they may not work; at worst you may get some very confusing results.

Fortunately, there are not many of these. I've included a dummy section header at the top of my C5EDT.INI file to remind myself of them.

Listing 5. Dummy reminder section in my C5EDT.INI file.

```
[Text Editor Configuration]
_version=43

[** Key Mapping ** < jcm]
The following keys are reserved by the Editor:
AltF, AltE, AltA
CtrlZ, CtrlX, CtrlC, CtrlV, CtrlA, CtrlI, CtrlJ, CtrlK, CtrlL, CtrlM, CtrlN, CtrlO, CtrlP, CtrlQ, CtrlR, CtrlS, CtrlT, CtrlU, CtrlV, CtrlW, CtrlX, CtrlY, CtrlZ, Ctrl[, Ctrl], Ctrl~, Ctrl^, Ctrl_
F3, AltF3, ShiftF3, CtrlF3

[Key Mapping]
CharLeft=LeftKey
```


So what can be safely changed? Well, I've made quite a few changes to my C5EDT.INI file to make the editor work a bit more like the way I want it to. For example:

The standard Clarion5 keyboard editor command to delete a word ahead of the cursor is `CtrlT` (goodness know why!). My fingers refuse to accept this; they keep using `CtrlW` no matter what! As you'll soon see, this was easy to change.

Step 3a. First have a look through C5EDT.INI to find the command most likely to do what you want to do. (This is a bit like trying to find the right embed point isn't it?) In my example, it's the `DeleteWord` command.

Step 3b. Now do a Find/Search through C5EDT.INI to make sure that the key combination you'd like to use is not already being used for something else. If it is, then you have two paths to take...

If you'd decided to reconfigure, say, `CtrlF` to some new purpose then you're out of luck. `CtrlF` is in that reserved list of key combinations I mentioned earlier (see Listing 5). It's reserved by the Clarion5 editor to enter `FormatStructure` mode (as you old CPD 2.1 users out there will remember well).

If, however, your new key combination is being used for some other purpose - but not in a special reserved manner - then it's a case of making a choice. If you like your own new reconfiguration better than the existing purpose defined for the key combination then it's just a matter of removing the existing definition.

NOTE: If you do this then you should also create a new key combination for the action which you "stole" from. (Sometimes, making a change can set off a whole chain-reaction of changes.)

Step 3c. Going back to my example, since `CtrlW` is not used for any other purpose it was a simple matter to change the original line from;

```
DeleteWord=CtrlT
```

to

```
DeleteWord=CtrlW
```

I have two options available for the use of `CtrlT` because it is now, potentially, freed-up by my change. Either I can reuse it for some other customised purpose (provided it's not one of the reserved key combinations - which it is!) or I can keep it as an alternative key combination for the same command.

To create two or more key combinations for the same command you just separate them with a semi-colon. For example;

```
DeleteWord=CtrlT;CtrlW
```

Now both these key combinations will achieve the same purpose. (In fact, even without including `CtrlT`, using this key combination will still delete words as before, because it's one of the special reserved key combinations that can't be redefined - see Listing 5).

Step 4. So, off you go - make the editor work more like the way you want it to, not the other way around.

Listing 6 shows selected contents of standard [Key Mapping] settings. Listing 7 show how I've chosen to redefine the same commands.

Listing 6. The [Key Mapping] section of the standard C5EDT.INI file.

```
[Key Mapping]
(Some lines skipped)
ScrollDown=CtrlUp
ScrollUp=CtrlDown
TopOfPage=CtrlPgUp
BottomOfPage=CtrlPgDn
TopOfFile=CtrlHome
EndOfFile=CtrlEnd
LeftOnNextLine=CtrlEnter
GotoLine=CtrlG
BackTab=ShiftTab
ToggleInsert=InsertKey
DeleteChar=DeleteKey
DeleteLeft=BSKey;ShiftBS
(Some more lines skipped)
StartMarking=
StartOfBlock=
WriteBlock=
CommentBlock=
CtrlChar=
DeleteBlock=
DeleteEOL=
FindReplace=
InsertLine=
NewChar=
ReadBlock=
```

Listing 7. The [Key Mapping] section of my C5EDT.INI file.

```
[Key Mapping]
(Some lines skipped)
ScrollDown=CtrlU
ScrollUp=CtrlD
TopOfPage=CtrlF1
BottomOfPage=CtrlF10
TopOfFile=AltHome
EndOfFile=AltEnd
LeftOnNextLine=
GotoLine=AltG
BackTab=ShiftTab
ToggleInsert=InsertKey
DeleteChar=DeleteKey
DeleteLeft=BSKey;CtrlBS
(Some more lines skipped)
StartMarking=CtrlEnter
StartOfBlock=
WriteBlock=F8Key
CommentBlock=
CtrlChar=
DeleteBlock=
DeleteEOL=CtrlEnd
FindReplace=AltC
InsertLine=CtrlI
NewChar=
ReadBlock=F9Key
```

It's very unlikely that you'll want to make exactly the same changes I've made, but hopefully these examples will get you started. (They will at least make you wonder about the logic in my fingers!)

[John Morter](#) is a member of the Victorian (Australia) Clarion Users Group. These days he's an IT Consultant in the corporate world who'd rather be spending all day building Clarion applications. Instead, Clarion occupies a lot of his spare time. John sails during the summer on his racing catamaran named Flat Chat, which is Australian slang for "at top speed!"

In This Issue

[The SQL Answer Cowboy](#)

Posted on July 13, 1999

[Larry Teames On Reports](#)

Posted on July 13, 1999

[Customizing Clarion5's Editor And Menus](#)

Posted on July 13, 1999

[Clarion News, July 1999](#)

Posted on July 13, 1999

[Photo Gallery](#)

Posted on July 13, 1999

[Publishing Schedule](#)

Posted on July 13, 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, email 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

Photo Gallery

- [The Raleigh Clarion User Group](#) (Posted July 12, 1999)

Send Us Your Photos!

One of the things I like best about Clarion conferences is the opportunity to put faces to names of people I've come to know through email or the newsgroups (or in a bygone era, CompuServe).

That said, there's no particular reason to wait for the next conference to see more ugly mugs. If you have a picture of yourself, your user group, or if you're a really lonely programmer, your favourite PC (just kidding – we really don't want any PC pictures), send them along. Photos from past conferences are also welcome. JPEGs or GIFs are preferable, and please include a caption or other descriptive material with each picture.

Send your favourite Clarion-related pictures to editor@clarionmag.com. Please indicate if you wish a copyright notice attached to those selected for publication.

In This Issue

[The SQL Answer Cowboy](#)

Posted on July 13, 1999

[Larry Teames On Reports](#)

Posted on July 13, 1999

[Customizing Clarion5's Editor And Menus](#)

Posted on July 13, 1999

[Clarion News, July 1999](#)

Posted on July 13, 1999

[Photo Gallery](#)

Posted on July 13, 1999

[Publishing Schedule](#)

Posted on July 13, 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, email covecomm@mbnet.mb.ca.



Clarion magazine

Publishing Schedule

Clarion Magazine is now published four times per month, on Tuesdays. This allows us to present you with up-to-date news items as well as a steady diet of Clarion programming information from the best writers in the Clarion community. As some months have five Tuesdays, publication occasionally skips a week.

In other respects, Clarion Magazine resembles a monthly magazine. Internally, the contents of the magazine are organized by month, and this is reflected in the [Site Index](#). When you purchase a year's [subscription](#), you have access to all of the articles for the month in which you begin your subscription, and for the next 11 months.

Main Page

Log In

Subscribe

Open Source

Links

Mailing Lists

Advertising

Submissions

Contact Us

Site Index

ClarionMag FAQ

Download PDFs

Search ClarionMag

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, email 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

Product Review

SearchFlash 1.1

Reviewed by Dave Harms

SearchFlash is a set of templates, procedures, and data files that make it relatively easy to add useful searching and tagging to any Clarion browse. The combination of searching and tagging is particularly appropriate since there are many situations where you want to restrict the display of records to just those which meet the search criteria.

I tested SearchFlash on the application I use for tracking Clarion Magazine subscriptions. The installation of SearchFlash went smoothly; the install program correctly located my Clarion5 directory (a nice touch), and asked me if I wanted to install the ABC or Legacy templates. (If you wish to install both you'll need to make sure you use a different demo directory each time.)

The help file, which can be displayed after install, offers specific instructions on installing SearchFlash. The steps are as follows:

Step 1. Register the template.

Step 2. Import SFLASH.TXD into your application's dictionary. This TXD contains definitions for files used to store tagging and query information. (There is a separate TXD containing just the tag file, and you can import this multiple times if needed, giving the file a unique name and prefix each time.)

Step 3. Close the dictionary, open the application, and import the seven core procedures from DEMO.APP. You'll want to follow the help closely to be sure you get just the procedures you need, as DEMO.APP contains a number of other procedures. A separate application containing just the core procedures might be a better way to go.

Step 4. Go to the Global extensions list and add the SearchFlash extension. You will need to add the files whose browses use the SearchFlash button.

Step 5. Populate the SearchFlash button onto the window containing the browse you want to be able to search. The help file again contains useful, specific information. A checkbox for filtering the browse to show only tagged items is also populated.

Step 6. Add a tag icon to your list box.

Step 7. The help specifies that you need to add SFLFirst, SFLNext and SFLSummary to the list of called procedures for that browse. I found it easier to ensure that those procedures have the Declare Globally box checked, thereby saving myself a few extra steps on subsequent browses.

Step 8. Fill in the SearchFlash button options (see Figure 1). At a minimum you'll need to specify the file SearchFlash is searching, a unique record ID field, the tag icon field, and some information for each tab on your browse including the key and key fields and any range limits.



Figure 1. SearchFlash button template prompts, Primary File tab.

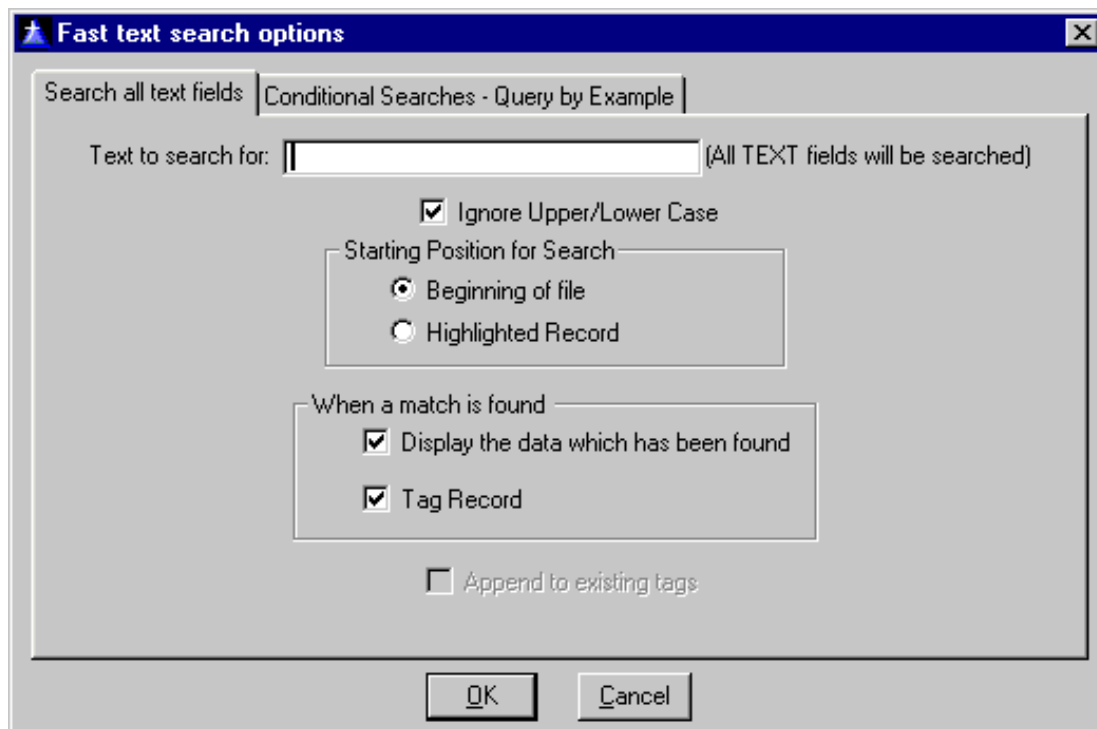
Warning: SearchFlash needs to know the name of your browse object, and it assumes that you've followed the ABC convention of numbering browse objects. If you've renamed your browse objects to something meaningful you'll just have to change them back again to BRWx, because all you're allowed to specify in the SearchFlash templates is the browse object number (see the Parent's Browse Ref prompt), not the name.

For browses with a lot of tabs it might be difficult to synchronize all of the search template settings with your browse's settings. It would be ideal if the search template were attached to the browse and could inherit the necessary information, but I don't know if this is a practical alternative.

Text Field Searching

Living dangerously, I tested SearchFlash on a browse which also uses Brian Staff's [Xplore templates](#) and the two products co-existed without any difficulties. I began by testing the "all text fields" search, shown in Figure 2.

Figure 2. Searching on all text fields.



Note that you can start searching from the beginning of the file or from the current position. If you choose to display the data as it is found SearchFlash will show you the record number, the name of the field in which it found the data (remember that on this tab you're searching all the text fields), and the data itself.

If you choose to tag records as they're found you can then use the Tagged Records checkbox to restrict the browse to only the found records. The number of tagged records is shown in the caption bar, and tags can be appended to existing tags so you can build up a result set from a series of searches.

SearchFlash also provides three ways of storing tagging information: in a global queue (the default); in a tag field in the file; or in a separate tagging file. For smaller data sets the queue is the fastest, though of course tagging information is lost when the application is closed.

QBE Searching

You can use QBE and text searching in conjunction if you wish, leveraging the power of both techniques. All of the tagging features described for text field searching apply to QBE searching as well. One difference is that in a QBE search the found data cannot not displayed for each record since you may be searching for multiple conditions.

Figure 3 shows the QBE tab on the search window. This tab lets you query up to six fields using AND or OR conditions. You can also save and retrieve your queries from this window.

Figure 3. Searching using QBE.

Fast text search options

Search all text fields | Conditional Searches - Query by Example

Join	Field	Operator	Value
	CUS:City	Contains	Boston
AND	CUS:Comments	Contains	newsgroup
AND		Contains	
AND		Contains	
AND		Contains	
AND		Contains	

Saved Queries

Save Retrieve Reset

OK Cancel

Additional Templates

Search flash comes with control templates for Tag All, Untag All, and tag toggle buttons. There are also templates to allow the use of tagged records in reports and processes.

SearchFlash is compatible with all versions from CW2002 to C5 ABC/Legacy, and all source code is supplied.

Clearly a lot of thought has gone into this product. Instructions on using the template in a multi-dll app are included (a most important feature) and there are a number of settings available for fine-tuning behaviour, such as setting fields to exclude from QBE searches and searching related files.

The help is quite clear and concise, and includes a FAQ to help the developer over common problems. I also received prompt response to several questions about the use of the product. The author, Mike McLoughlin, is active in the TS newsgroups.

Minor quibbles aside, this is an excellent tool which will enhance your users' ability to locate and display data.

Price	\$149
Web Site	http://www.sterlingdata.com/search.htm
Contact	Mike McLoughlin

In This Issue

[Product Review: SearchFlash](#)

Posted on July 20, 1999

[The Other Way To Use OLE](#)







Posted on July 20, 1999





[The Clarion Advisor: Debug Redux](#)

Posted on July 20, 1999

[Clarion News, July 1999](#)

Posted on July 20, 1999

PRODUCT RATING	
Documentation	
Features	
Support	
Ease of use	
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, email 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

Calling OLE Methods - Part 1

by Jim Kane

One day a long time ago when CompuServe and not the newsgroups reined supreme, someone asked how to create a shortcut using Clarion. After a quick trip to my trusty [MSDN](#) (Microsoft Developer Network) CD, it became apparent to me that to create a shortcut I had to call an OLE interface called IShellLink. Unfortunately, while Clarion 5 supports the Topspeed, Pascal, and C calling conventions, it does not natively support calling OLE interfaces.

There are two general methods of calling OLE objects: late binding and early binding. Most OCX and OLE code Clarion programmers are use to seeing is late -binding, where the address of each method is looked up via the Idispatch API call before each call. No compile-time knowledge of the order the methods are in is needed for late binding; only the method name is required.

The early binding discussed here a more efficient approach, although it requires that you know the number of methods and the order in which they appear in the OLE interface. The address of the area which contains the method entry points is determined once and stored, and as all of the methods are at a fixed offset from this point they can subsequently be called based on that offset.

While calling a method based on its offset from a known address is slightly less efficient than the normal way of calling an API function directly, the extra step is quick and saves the trouble of storing the address for each method separately.

The Keys To The Kingdom

I posted an answer to the original question about creating a shell link that in effect that could have been summed up in two words: "No Way." Never liking to accept limits, when I next had some free time (about a year later), I read all about how to call OLE interfaces. I realize that with a little straightforward assembler work I could write one procedure (or a family of procedures) that would handle all the dirty work for me. Once you understand this one little ugly piece and how to use it, you'll be able to do anything Windows can do. Sorry, but no more excuses then!

My reading showed that a call to the API CoCreateInstance procedure returned the keys to the kingdom, so to speak, or at least a pointer to a pointer to the entrance to the OLE world. If only I could take that pointer to a pointer and call the address at the end of the chain, OLE would be mine.

Actually what CoCreateInstance returns is a pointer to a pointer to a table of addresses, called a vtable. A vtable is a simple list of addresses of all the methods in an OLE interface. For those not using OOP think of a method as a procedure. An OLE Interface can be thought of, for this discussion, as a collection of methods or procedures that when called does something. It's actually a whole lot more but that will do for now.



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



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

Finding The vTable

Calling an OLE interface is a two step process. First you navigate the pointer chain to get to the vtable, and then you determine where in the vtable the procedure to be called is located.

I'll take a common OLE interface called Iunknown (which also pretty well sums what I've said about OLE at this point!) with three methods: QueryInterface, AddrOf, Release. For now don't worry about what the methods do. Listing 1 shows what the mystical vtable would look like in Clarion syntax:

Listing 1. A vtable declared in Clarion code.

```
Vtable_IUnknown Group
AddressofIUnknown_QueryInterface      long
AddressofIUnknown_Addref               long
AddressofIUnknown_release              long
end
```

Now I'll diagram the chain of pointers to arrive at a vtable so it's not so abstract. I'll store the pointer to a pointer in a Clarion long variable called ppVtable (pp has nothing to do with the bathroom in this case, it's a shorthand for pointer to a pointer) and the offset into the vtable in a long variable called ofsMethod. Figure 1 uses some arbitrary addresses to demonstrate how to get to the code for the Iunknown.AddrOf method. Since AddrOf is one down the list from the top and each long is four bytes, it's address is stored at the start of the vtable plus an offset of 4H.

Figure 1. The vtable data.

<u>Arbitrary Address</u>	<u>Variable Name/Description</u>	<u>Contents</u>
120H	OfsMethod	124H
124H	ppVTable	58204H
58204H	pVTable - pointer to the vtable	7F800H
7F800H	Vtable - Address of 1 st method (AddressofIUnknown_QueryInterface)	9AB00H
7F804H	Vtable - Address of 2 nd Method (AddressofIunknown_AddRef)	9C000H

At address 9AB00H sits the executable code for Iunknown.QueryInterface.

At address 9C000H sits the executable code for Iunknown_AddRef.

So given a pointer to a pointer for a vtable (in this case the value 58204H), if you look in memory at that location you get the pointer to the vtable (or pVTable) at 7F800H. If you look in memory at 7F800H you find (at long last!) the start of the vtable. Jump down to an offset of four bytes and look in memory at 7F804H and you're all set: 9C000H is the correct target address. This means lots of peeks and jumping around but it's a relatively simple chain to follow. The chain navigation code only has to be written once, and after that's done hopefully you can get a lot of good use out of it.

Five Steps To OLE

Thinking further, the problem of calling an OLE interface (after obtaining a pointer to a vtable) really has five parts (one of which I've kept a secret until now):

1. Put the parameters for the OLE method, if any, on the stack (I'll explain the stack in a moment).
2. Put the pointer to a pointer to the vtable on the stack – a requirement of calling an OLE Interface (this is the new thing I held back on previously).
3. Put the return address to your Clarion code on the stack.
4. Navigate the pointer to a pointer bit to arrive at the vtable.
5. Once at the vtable, apply the offset for the method you want and jump/call to that address.

I'm visually oriented so I'll turn that into a picture. A stack is an area of memory, just like a stack of books where each "book" is 32 bits. Every time you push something onto the stack it goes on top. Every time you pop something off the stack, a book goes away and exposes the book underneath. A stack grows from high memory to low memory. Here's what the stack looks like in a simple procedure call (not an OLE interface) and diagram:

```
Module('SomeDll.Dll')
  AddTwoNumbers(long p1, short p2), long, pascal
End
```

Code to call:

```
Result = AddTwoNumbers(2,3)
!After call to AddTwoNumbers
Message('Result is:' & Result, 'Result')
```

Procedure code:

```
AddTwoNumbers procedure(long p1, short p2)
Result Long
Code
!Start of procedure
Result = p1 + p2
Return Result
```

Here's what the stack looks like at the comment !Start of procedure (the absolute address values are arbitrary):

<u>Address</u>	<u>Stack Contents</u>
1F4H	ReturnAddress - address of Message statement
1F8H	p1=2
1FCH	p2=3

Notice that although P2 is a short, on the stack it takes up 32 bits from 1F8H to 1FBH. In fact, all parameters on the stack take up 32 bits no matter if the value being passed is a byte, short or long! If a parameter is longer than 32 bytes, such as a cstring, then rather than trying to put the entire cstring on the stack, just its address is placed on the stack.

After the execution of the Return Result statement, the three values shown above are removed from the stack. This is how the Pascal calling convention works. All OLE interfaces use the Pascal calling convention as do almost all API calls.

To complete the picture look at the assembler code to call AddTwoNumbers. It's not very complex:

```

Mov  eax, p2          ! move p2 into the eax register
Push eax             ! Push  a long onto the stack in this case
                    ! p2 with a value of 3
Mov   eax, p1        ! put p1 into eax, want to guess where its
                    ! going next?
Push  eax            ! p1 on the stack
Call  AddTwoNumbers  ! puts the return address on the stack
                    ! and jumps to the start of the procedure
                    ! AddTwoNumbers.

```

After that entertaining jaunt into the world of assembler and the stack lets get back to the problem at hand. Say you have a pointer to a pointer to a vtable for an OLE interface call `IshellLink`. You'll store that value in a long called `ppVtable_IshellLink`. Further, say `IshellLink` has a method call `SetPath` that can be called with one parameter, the address of a `cstring`. The purpose of the method is to set the path and file name for the target of a shortcut, i.e. the program that should be run when a shortcut is clicked. If Clarion directly supported OLE, you might think of this as calling a class method and would prototype it something like this:

```
IshellLink.SetPath(*cstring szPath),Pascal
```

and call it (if it worked!) to take a step to creating a shortcut to notepad like this:

```
SzPath = 'C:\Windows\notepad.exe'
IshellLink.SetPath(szPath)
```

Since `*cstring` is a pointer to a `cstring`, or in other words the memory address of the `cstring`, you could also prototype and call the above like this with the same result:

```
IshellLink.SetPath(long pszPath),Pascal
```

And use the `address()` function to get the address:

```
IshellLink.SetPath(address(szPath))
```

Keep in mind Clarion does not support this calling convention so the above is hypothetical.

If you review the requirements for calling an OLE interface listed above, the picture you need on the stack for this to work is:

Figure 2. Stack data for calling OLE interface

Arbitrary Address	Value or Description
200H	Return Address to Clarion code after the call to the OLE Interface
1FCH	ppVtable
1F8H	address(szPath)

The other piece of information you need to be able to jump to the OLE method of choice is the offset to the `SetPath` method. You'll need to put that on the stack as well.

It's looking like to call an OLE interface with one parameter what you need is a magic function called

```
ICall1P(long ppVtable, long ofsMethod, long p1),long, pascal,proc
```

that can create a stack like the one shown above and jump to the correct vtable entry.

Now you can write the Clarion code part so you can picture it better, diagram the stack this magic procedure will start with, and lastly present the assembler code to accomplish what you need.

```
Module(ICall.a)
  ICall1P(long ppVtable, long ofsMethod, long p1),long,
    pascal,proc,Name('ICall')
End

Data:
SzPath      cstring('C:\Windows\notepad.exe')
PpVtable    long(58204H)      !pointer to a pointer for the
                                ! vtable for IShellLink.
ofsMethod   long(50H)        !offset to SetPath
Hr          Long,Auto        !Ole Return code, < 0 = error

Code:
Hr = ICall1P(ppVtable, ofsMethod, Address(szPath))
If Hr<0 then
  Message('Call Failed, blame Bill')
else
  Message('Call Worked, blame Jim Kane')
end
```

After the calling `ICall1P` this is what the stack will look like:

```
ReturnAddress = address of If Hr<0 after ICall
PpVtable
ofsMethod
P1
```

If you compare that to the desired stack frame above, you will see the extra `ofsMethod` in there. Also there are some mundane details of preserving registers. You can think of a register in the CPU as if it's a 32 bit variable. The registers you will deal with are `eax`, `ebx`, `ecx`, `edx`. The code to get the stack from the starting point just above to the desired status prior to turning things over the OLE and Microsoft is as follows:

```

Public ICall:
(*The parameters, how ever many there were are on the stack*)
(*code to save ebx,ecx,edx-omitted for clarity *)
  Pop ecx          (* pop the return address off the stack *)
                  (* into ecx *)
  Mov Save_ret,ecx (* save the return address for later *)
  Pop eax          (* eax = ppVtable*)
  Pop ebx          (* ebx = offset into vtable*)
  Push eax         (* put the ppVtable back onto the stack *)
                  (* now that the offset is out of the way*)
  Mov ecx,[eax]   (* ecx = contents of memory at eax = *)
                  (* pVtable and NOT ppvtable any more *)
  Add ecx,ebx     (* add in the offset down the vtable*)
                  (* ecx now points to the address you *)
                  (* want to call*)
  call dword [ecx] (*put the return address on the stack*)
                  (* and go to the address pointed to by ecx*)
                  (* upon return eax = the return value so leave eax alone! *)
                  (* the return from the OLE method also took the *)
                  (* parameters pl...pn off the stack.*)
  mov ecx, Save_ret
  push ecx        (* put the return address to the Clarion *)
                  (* calling point back on the stack *)
                  (* code to restore ebx,ecx,edx- omitted for clarity*)
  ret 0           (* I love it when a plan comes together!*)

```

You're Doing OLE!

So there you have it. Add those few lines of assembler to your code and OLE away just like the big boys. Actually just choose Project from the main menu and add Icall.a to the external source module section, add the prototypes for Icall1P to the global map. The project system will take it from there - the assembler will be called to assemble Icall.a and the Clarion compiler will compile the rest. The project system recognizes the portion that needs to be assembled by the .a extension.

The nice thing is regardless of how many parameters the interface method has, the code above should handle the details of calling the interface. It requires just two inputs: the ppVtable available from CoCreateInstance and the offset into the vtable available from OLE header files or in some cases from a free Microsoft utility called OLEView. Now you have reduced the barrier to COM in Clarion to finding the needed constants and preparing the interface method parameters. Next time I'll take on those challenges and show how to create a shortcut the OLE way.

So if you catch me on the newsgroups, feel free to ask a question. Just remember it may take a year or more to get an answer!

[Download the source code](#)

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

In This Issue

[Product Review:
SearchFlash](#)

Posted on July 20,
1999

[The Other Way
To Use OLE](#)

Posted on July 20,
1999

[The Clarion
Advisor: Debug
Redux](#)

Posted on July 20,
1999

[Clarion News,
July 1999](#)

Posted on July 20,
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, email 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 Advisor

In an [earlier Clarion Advisor article](#) I looked at some alternative debugging techniques and asked readers if they had any tricks to share. They did, and I am again impressed by the inventiveness of Clarion developers.

Both Russ Eggen and Carl Barnes like to use INI files and the clipboard, though their approaches differ. Russ also offers a nifty technique for fixing SQL problems, and Carl explains how to use file logging to good advantage.

Russ Eggen

If you are processing (reports, process and browse populate functions) and need to find out what is going on behind the scenes, make your own dump log. You can use PUTINI (or use the ABC INIMgr) to write out your own statements. This make is far easier to look through the items that you are interested. And if you use the Defines tab on the project setting (to conditionally compile your code – ed.), you can turn this on or off as needed and not worry about stripping out the debug code when done.

If you are using SQL statements and the last statement comes back from the database with some error, it typically means that there is a syntax error on your part. Before sending the command via PROP:SQL, copy it to the clipboard. Since the clipboard has only the last thing sent to it, you can open up you favorite SQL command editor, paste and try to compile or execute (depending on tool used). This will expose the syntax error (like unknown column name in the case of misspellings, missing a closing quote, etc).

Carl Barnes

My favorite is to use PUTINI to write info I need to a text file. It's easy to code and since the file is kept closed you can display your information most any time. Plus PUTINI offers a lot of options in the way you can structure your information into sections, such as logging all the events.

```
ACCEPT
  evt += 1
  PUTINI('Events',evt,'Event=' & event() |
    & ' Field=' & field() ,'. /0debug')
END
```

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!

Another option is to use `SETCLIPBOARD()` to put some information you want to know in the clipboard. Then you can use the Clipboard viewer to watch for changes. This does have the limitation of only showing the last entry and not a log. There are some clipboard savers that will retain previous clipboard contents ([PC Magazine](#) has a number of clipboard viewers available for download).

The Clarion file drivers support logging activity to a text file. There is more on this in the C5 Programmer's guide (see "Logging"). Logging can be turned on for all files by adding entries to your Win.INI file. This is useful when a user has some sort of crash happening during start up and you're not sure what file is the problem.

```
;add to Win.Ini
[CWdriver]           e.g. [CWTopSpeed]
Profile=[1|0]        1=on, 0=off
Details=[1|0]
Trace=[1|0]
TraceFile=[Pathname]
```

There is also "On Demand" logging that can be turned on and off using the property syntax. You can even add your own comments to the log file to help you know what your code is trying to do. This is useful when some complex file access code is not working and you would like to see what it is really doing. For example I had to write code that synchronizes one file with two others.

```
Myfile{PROP:Profile}=' \MyFile.log' !Turns Clarion I/O logging on
Myfile{PROP:Details}=1             !Turns Record Buffer logging on
! Write the string to the log file
Myfile{PROP:Log}='Starting Big Post ' & clock()
DO BigProcessRtn
Myfile{PROP:Details}=0             !Turns Record Buffer logging off
Myfile{PROP:Profile}=''           !Turns Clarion I/O logging off
```

Be sure to try logging and turn on all the options to see the amount of data you can get about file access. I turn it on now and then to see what is happening during the startup of my programs. Don't forget to turn it off; the log files can take a lot of space and slow things down.

In This Issue

[Product Review:
SearchFlash](#)

Posted on July 20,
1999

[The Other Way
To Use OLE](#)

Posted on July 20,
1999

[The Clarion
Advisor: Debug
Redux](#)

Posted on July 20,
1999

[Clarion News,
July 1999](#)

Posted on July 20,
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, email 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

The ABC's of Control Files

by Steve Parker

My very deep appreciation to Nik Johnson who bailed me out of control file purgatory (but you know what I really meant) when making the transition to ABC. The solution is his; the text of our exchange is at the end of this article.

I've been using control files since... I can't remember when. When I began moving to ABC, I was confident. My experiences with the app converter had been by and large quite good. But the code that accessed and maintained my control files didn't work after conversion.

Consulting Richard Taylor's superb "Making the Transition to ABC" in the online help, I was able to make some adjustments to what the converter had done. Still, it didn't work.

Control files, clearly, had become a different breed of cat in ABC.

What Is A Control File? And Why Should I Use One?

A control file has two unique characteristics. First, a control file has only one record. Second, a control file has no keys (with only one record, a key is sort of pointless).

In many respects, control files duplicate the function of INI files. The important difference is that a control file, unless you use the ASCII or Basic driver (bad move), cannot be read with a text editor, is not so easily corrupted or manipulated as a text file and, should you so wish, can be encrypted. All or part of a standard file, which is what a control file is, can be made read-only.

The purposes for which you use or, indeed, whether you use a control file as opposed to an INI file is, of course, entirely up to you. But if you decide use a control file, how you use it is not.

The Issues

Just as with any file, there two kinds of things you will want to do. First, you'll want to write to the control file, adding and updating records... oops, the record. When adding, of course, you'll want to add only to an empty file. Adding a second record to a control file defeats its purpose. So, adding is a "one time thing."

Second, you'll want access the record, i.e., read the file. Because a control file has only one record, by definition, and no keys, none of the standard template methods of file handling will work as expected. Indeed, they will not work at all.

Sequential processing is not possible (no "sequence," you see). There is no key to prime. `Set(key)` and `Set(key,key)` have nothing to operate on. A loop, of course, is pointless. In plain English, the standard templates can do nothing for you except open and close the file.

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

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

Similarly, you can't simply add records to the file. A control file may have one and no more than one record. The standard templates will try to add multiple records.

The problem is determining (1) whether or not the file exists at runtime, (2) whether or not there is in fact a record in the file and (3) how to tell a file what you want to do (Add or Put).

(1) is usually only a problem the first time the app is run.

To make matters more interesting, the database driver that you use makes a difference in accessing single record files. TPS files, in particular, do not support the `Pointer()` function for direct record retrieval while most other non-SQL databases do. Your choice of file systems will make a difference in your handling of control files.

Accessing Control Files

For the moment, let's assume that there is already a record in the control file.

If you want to display the user's company name and address in the header of a report or plug the city, state and postal code on data entry form, the data contained in a record must have been successfully read first. If you haven't first read the record:

```
CUS:City = CFG:City
```

isn't likely to give the desired result, is it?

In CW 2.0, or even in DOS, you would do something like the following:

```
Open (file)
Get (file, 1)
```

Take a moment to look at this code.

First, the file is opened. This does nothing but create a record buffer in memory. Nothing here prepares the file to be read. Ok, it's not "nothing."

Second, normal file handling would follow with a `Next()` or `Set()/Next()`. But a control file contains only one record. If the driver fully supports `Pointer()`, the record can be accessed directly,

```
Get(file,1)
```

The lesson? (1) Open, (2) read. Read = retain for later use.

TPS files don't fully support the `Pointer()` function (on a TPS file, `Pointer()` returns a valid pointer which can be used for direct retrieval but "1" is not a valid pointer with TPS files), so:

```
Open(Config,42h) !or Share()
Set(Config)
Next(Config)
```

is required for TPS files.

Legacy command such as `Open()` and `Get()` can in fact be used in these circumstances even in ABC. Bad form, to be sure. But they will work. You must ensure that the file is closed at the appropriate point, if you open the file directly.

In ABC, "good form" would be:

```
Access:Config.Open
Set(Config)
Access:Config.Next()
```

for TPS files.

`Access:file.Open` opens the file. `Set()` prepares the file for reading, as always. And `Access:file.Next()` reads the first (and in this case only)

record, if any.

You will notice there is no ABC analog for the `Get()` command. Therefore, for file systems fully supporting `Pointer()`, you can continue using `Get()`:

```
Access:Config.Open
Get(Config,1)
```

With these file systems, xBase, Clarion, etc., I have had the `Set()/Next()` strategy fail. Thus, I continue using `Get()`.

Maintaining Control Files

Since there is only one record, a browse is sort of... ah, pointless. Go directly to the form.

The problem is that a form needs to be told what to do. `GlobalRequest` is typically used to tell the form whether it is being call to add, update or delete a record. Without a legitimate value in `GlobalRequest`, the form won't do anything.

Specifically, if called to add a new record, the form needs an empty buffer. If called to change or delete, the form needs a buffer with the correct record.

So, `GlobalRequest` needs to be set; what's the big deal? The big deal is that you need to know whether or not the file has a record before you set `GlobalRequest`. If there is no record, `GlobalRequest` must be `InsertRecord` and if there is a record `GlobalRequest` must be `ChangeRecord` (you don't ever want to delete a control record, do you?).

As it turns out, this is easily determined. If `Relate:file.Open` returns a non-zero value, there was an error opening the file (e.g., the file does not exist and is not set up for create-if-not-found). If `Access:file.Next()` returns a value, there is no record in the file. So the following code can be used to set up the call to a form:

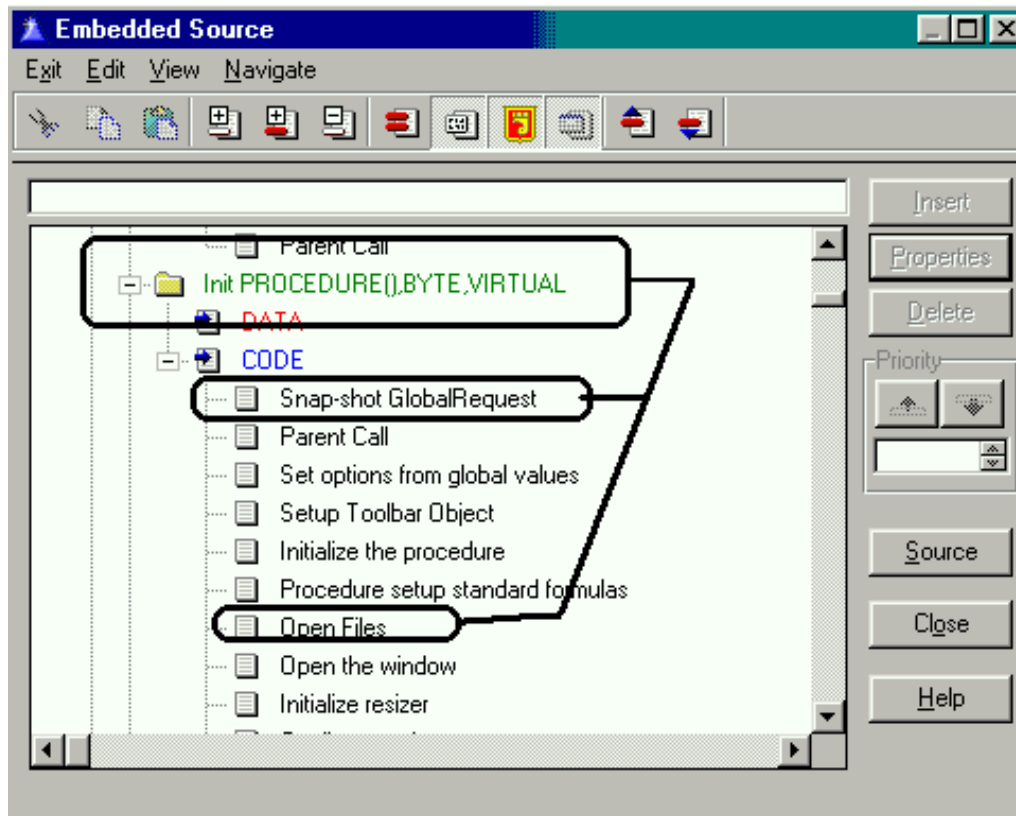
Listing 1. Setting up the call to the control file form.

```
IF NOT Relate:Config.Open()
  SET(Config)
  IF Access:Config.Next()
    GlobalRequest = InsertRecord
  ELSE
    GlobalRequest = ChangeRecord
  END
  SetupForm
  Relate:Config.Close
Else
  !action if file create not allowed
END
```

This ensures that `GlobalRequest` is always set properly.

Trying to determine the appropriate value for `GlobalRequest` from inside the Form procedure is a bit more difficult. It is more difficult because `GlobalRequest` is read before the file is opened (see Figure 1), early in the `INIT` method.

Figure 1. The Init method embed points.



Of course, the code in Listing 1 will work perfectly well before "Snap-shot GlobalRequest" (without the procedure call, of course) and not wreck havoc on the standard template code (so long as you close the file first).

To ensure that the configuration file is properly opened and read when the program is started, I use something like:

Listing 2. Opening the configuration file.

```

Loop
  Access:Config.Open
  Set(Config)
  If Access:Config.Next()
    SetupForm
  Else
    Break
  End
End

```

in the main procedure's INIT method. Notice that this code loops until there is no problem opening and reading the file. Combined with the code in Listing 1, the form always knows what is required of it.

Summary

Config files aren't especially difficult. But they do prove just how spoiled we are. Everywhere else, Clarion sets up the file buffer, accesses files and sets up forms. When there's only one record in the file, you're on your own as far as these go.

On the other hand, if you can manipulate a configuration file, you know how to access a file.

In next week's issue of Clarion Magazine, Nik Johnson takes control files

In This Issue

[Working With Control Files I](#)

Posted on July 27, 1999

[Clarion Challenge String Parser](#)

[Final Results](#)

Posted on July 27, 1999

[Clarion News.](#)

Listing 3. Nik and Steve's correspondence.

```
> I have a single record (configuration) file and, based on what
> I did in CW2003, I used:
>
> Access:Config.Open
> Set(Config)
> Access:Config.Next()
>
> to get that record, but it doesn't seem to be returning the
> correct >field values.Could it be that the error is in my file
> access in the update form:
>
> Access:Config.Next()
> If ErrorCode()
>   If ErrorCode() = 35 then ThisWindow.Request = InsertRecord.
> Else
>   ThisWindow.Request = ChangeRecord
> End
> ThisWindow.OriginalRequest = ThisWindow.Request
>
> because I noticed (finally) that I had 15 or 16 records in the
> file. Ok, where'd I foul up? (Please...)
>
> TIA
>
> Steve Parker
```

(Nik Johnson's reply:)

I think you have to be careful about timing here. What I would do is have a source procedure which sits between the menu (or whatever triggers the configuration file update process) and the update form:

LinkUpdate PROCEDURE

```
CODE
IF NOT Relate:Config.Open
  SET(Config)
  IF Access:Config.Next()
    GlobalRequest = InsertRecord
  ELSE
    GlobalRequest = ChangeRecord
  END
ConfigForm()
Relate:Config.Close
END
RETURN
```

This makes sure that the file is open, the record is current (or the buffer cleared) and GlobalRequest is set -before- you ever get to the update form. In that way you don't have to worry about little things like the fact that the update window will internalize GlobalRequest before -it- opens the file.

The update form at this point should see no difference between this call for a singular record and a request from a browse of many records. Objects are like people. They function best when given directions in a familiar context.

-Nik

[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, email 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

String Parser Challenge Results

by Dave Harms

In a previous issue Clarion Magazine [issued a challenge](#) to Clarion developers to write some object-oriented code to parse strings. The five respondents listed in Table 1 made it through to the final evaluation, and Carl Barnes submitted the winning entry. All timings were done through 10,000 iterations of the test strings on a P233 laptop running NT4 SP3, in 32 bit, with debug off.

Name	Test 1	Test 2
Carl Barnes	7.1	8.7
Phil Will	10.3	n/a
Gordon Smith	11.0	14.6
Jesper Lorentzen and Maarten Bijl	12.9	13.6
Chris Hargett	14.7	n/a

Test 1 was included in the example application provided to all participants, and simply consisted of an English phrase which would be parsed by code written by the participants. The test then alternated the words between upper and lower case and wrote them back to the string. The code which performed the test is shown in Listing 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!

Listing 1. The first test (English text).

```

ParserBaseClass.Test                                     procedure
TempString string(200)
x               long
  code
  self.Reset()
  self.AddDelimiter(' ')
  self.SetString('This is the test string, which should have '|
    & 'its words alternating between upper case and lower '|
    & 'case. The actual test will parse Clarion code and '|
    & 'capitalize keywords.')
  self.BeforeTest()
  loop x = 1 to self.GetTokenCount()
    TempString = self.GetToken(x)
    if x % 2
      TempString = upper(TempString)
    else
      TempString = lower(TempString)
    end
    self.PutToken(x,TempString)
  end
end
return

```

The `Test` method in Listing 1 is straightforward. The `Reset` method clears any current text in the parser and removes any existing list of delimiters (which are strings used to separate words or "tokens"). The test then adds a single space character delimiter and sets the string the parser will parse.

The `BeforeTest` method is a placeholder virtual method which allows participants to call their own code from within the test method, much the way embed code is added to a Clarion application. Typically this is where the string is actually parsed. The `Test` method then loops through the string's tokens and alternately sets the case to upper or lower.

As I indicated in the initial challenge, there was a second test which the contestants did not receive, and which involved parsing Clarion code. That test is shown in Listing 2.

Listing 2. The second test (Clarion code).

```

ParserBaseClass.Test2                                   procedure
TempString string(200)
x               long
  code
  self.Reset()
  self.AddDelimiter(' ')
  self.AddDelimiter('.')
  self.AddDelimiter('&')
  self.AddDelimiter('(')
  self.AddDelimiter(')')
  self.AddDelimiter('=')
  self.AddDelimiter(',')
  self.AddDelimiter('-')
  self.AddDelimiter('+')
  self.SetString('self.Text=sub(self.Text,1,self.TokenQ.'|
    & 'Start-1)&clip(Text)&sub(self.Text,self.TokenQ.'|
    & 'Finish+1,Len(Self.Text))')
  self.BeforeTest()
  loop x = 1 to self.GetTokenCount()

```

```

TempString = self.GetToken(x)
case upper(TempString)
of 'SELF'
orof 'SUB'
orof 'CLIP'
orof 'LEN'
  TempString = upper(TempString)
  self.PutToken(x,TempString)
end
end
return

```

Test2 follows the same approach as Test but adds additional delimiters to enable the parser to pick out Clarion keywords. The test string (obtained from Jesper Lorentzen and Maarten Bijl's entry) contains four different keywords, so a simple Case statement is sufficient is all that's needed. Interestingly there are also no spaces used as delimiters, which means that no "lucky" parsing can happen (i.e. the parser detects the string (sub and the test code changes this to (SUB, with the UPPER having no effect on the parenthesis character.)

Although all the entries passed the first test, two had problems with the second, and those times aren't shown.

On the whole the difference between the fastest and slowest code can't be considered order-of-magnitude dramatic, but there were considerable differences in the participants' implementations. Chris Hargett and Phil Will both parsed the string on the fly, while the other three entries used the BeforeTest method to do a one-time parse of the string. There were some novel approaches to storing the delimiter data, and a number of variations possible on the implementation of PutToken. As Carl Barnes noted the ability to change a token's size complicates the matter somewhat. I declined to make this part of the test.

My thanks to all of the participants. This one's been somewhat grueling because of the complexity of the requirement and the amount of code that had to be written. As a result future challenges will revert to the original goal of not requiring more than a few lines of code, if still a bit of thought.

[Download the source](#)

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, email covecomm@mbnet.mb.ca.

In This Issue

[Working With Control Files I](#)

Posted on July 27, 1999

[Clarion Challenge String Parser Final Results](#)

Posted on July 27, 1999

[Clarion News, July 1999](#)

Posted on July 27, 1999



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

July 27, 1999

[Imaging Templates v1.05 Now Available](#)

This version fixes a few bugs and adds some additional Print Functions. Included changes are support for Group 4 TIFF files, fixes to the printer dialog and function, new prompt for initial zoom method, printing images without opening the image viewer, and printing via the OCX.

[Icon Locator/Viewer Source Code](#)

Sterling Data has just released as freeware the source code for Icon Boss, an icon locator and viewer. Full source code (C4/5 legacy) is available on the website.

[Dev Monitor Available Through TopSpeed Accessory Program](#)

Stealth Software's Dev Monitor product is now available on the TopSpeed Accessory program. Dev Monitor is tool for tracking development time and assisting in project estimating. Price is US\$199.00, and a [free download](#) of a Lotus Screencam is available.

[ABC Free Templates and Tools Updated July 23, 1999](#)

An updated version of the ABC Free Templates and Tools is now available for download. New features include ABC compliant file opening for CPCS reports, support for SendMessage API call, an enumeration template enhancement, form wizard improvements, and more.

[Activity Logging And Undo Template](#)

Sterling Data has released LogFlash, an activity logging template which also offers undo/undelete capability. Includes an extension template for update forms which logs file changes on a field by field basis. Inserts and deletes are also logged, and changes to individual fields can be rolled back. Source code included, ABC compatible. \$195

July 20, 1999

[C5B Wizatron-Compatible Droplist Query Template](#)

The DropQBE template is a query system compatible with Clarion5B which can be used in conjunction with the Clarion5 QBE Browse. Queries can be stored in a table and edited, recalled for later use, sent to a report etc. Also supports Larry Teames CPCS Report Tools and C5B Wizatron-generated procedures. A demo is available.

[Product Scope 32 Bookmarks V3.8 Officially Released.](#)

Product Scope 3.8 has been released. To download a full install or upgrade-only version (you must have the full V3.1 or V3.2 or V3.5 install to use the upgrade) visit the Encourager Software Web Site

[Win A Free Tropical Weekend Getaway!](#)

All attendees of the DevCon '99 Key West Fest are eligible to win a free two-night getaway at [Hawk's Cay Resort and Marina](#). These exclusive accommodations are just a short drive from Key West and offer snorkeling, diving, deep-sea fishing, jet skiing and more. The winner will be announced during the DevCon wrap-up session on Wednesday, September 29.

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

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

In This Issue

[Working With Control Files I](#)

Posted on July 27, 1999

[Clarion Challenge String Parser](#)

[Final Results](#)

Posted on July 27, 1999

[Clarion News, July 1999](#)

Posted on July 27, 1999

[SearchFlash 2 for 1 Special Offer](#)

As a special promotion for the SearchFlash templates Sterling Data is offering two templates for the price of one. Buy SearchFlash and receive a free copy of the CopyFlash templates. Offer ends July 23, 1999.

July 13, 1999

[QuickBooks/Quicken Developers Kit](#)

Intuit Australia now has developers' kits for QuickBooks and Quicken. Kits are listed at \$195 (Australian currency, presumably) and include documentation and interface tools. Word is the kit works with all international versions of Quicken and QuickBooks.

[Topspeed Updates Web Site](#)

Topspeed Corporation has given its web site a facelift. Drop by and see what you think!

[C5B Available For Download](#)

C5B is now available from TopSpeed's ftp site. Should you experience any difficulty with the download process, please contact TopSpeed sales department at (800) 354-5444.

July 6, 1999

[New GCal Build Available](#)

A new build for GCal is now available for download. This build fixes a couple of minor problems, check help file for details. A preview of the upcoming GCalPro release is also available.

[Mitten Software Offers OOP/API/SQL Training](#)

Mitten Software is now offering a series of training sessions with Intermediate to advanced level topics at regional locations around the country. The first few lessons include Randy Goodhew covering OOP and API and Andy (Cowboy) Stapleton teaching SQL and Clarion. The Clarion Certification Exam is an option at the Minneapolis location.

[Phone, Address, Postal Code Information](#)

Sterling Data has a web page for country format information including phone numbers, addressing, abbreviations, and postal codes. Contributions for countries not yet listed are welcome.

[Read the June 1999 News](#)

Do you have a news story or press release we should know about? Send it to 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, email covecomm@mbnet.mb.ca.