

Clarion MAGAZINE

Clarion
Development
Resources

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.

\$6.²⁵/month

[Main Page](#)

[Log In](#)
[Subscribe](#)

[Frequently Asked Questions](#)

[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To Other Sites](#)

[Downloads](#)
[Open Source Project](#)
[Issues in PDF Format](#)
[Free Software](#)

[Advertising](#)

[Contact Us](#)

Issue Index

December, 1999

[About Our New Look](#)

Change is good! Dave Harms talks about Clarion Magazine's new look. (Dec 7, 1999)

[Special Report: ConVic '99 Clarion Conference](#)

Simon Brewer details the happenings at this year's Clarion conference in Victoria, Australia. There's some amazing stuff happening down under. With pictures. (Dec 7, 1999)

[Template Writing Made Easy](#)

If you're not writing templates, you're missing out on a huge productivity gain. And it's easier than you think. (Dec 7, 1999)

[The SQL Answer Cowboy](#)

Andy Stapleton, the acknowledged Clarion SQL guru, answers your SQL questions. (Dec 7, 1999)

[The Clarion Advisor - Listbox Styles](#)

The Style checkbox in the listbox formatter shows promise, but the implementation falls short. Mike Pickus explains how to make listbox styles work with the templates. (Dec 7, 1999)

[The Cranky Programmer: Nits And Bits](#)

Cranky gets wound up about interface teases and embed error handling. (Dec 7, 1999)

[ABC Design Series: View Manager Part 2](#)

In ViewManager Part 2 David Bayliss looks at the code that actually does the work. (Dec 7, 1999)

[New ClarionMag Link Buttons](#)

If you'd like to link to Clarion Magazine, or you already have a link, be sure to get an updated link button to go with our new look. (Dec 7, 1999)

[ClarionMag Newsgroup Reminder](#)

We like to hear from readers. If you'd like to offer suggestions, ask questions about articles, or just chat with other subscribers, check out the private Clarion Magazine newsgroups. (Dec 7, 1999)

[Free Utilities: Class Browser And Compile Manager](#)

If you've been looking for Gordon Smith's Clarion Class Browser and his Compile Manager, you can now find them here.

TopSpeed

Clarion 5
by TopSpeed



(Dec 7, 1999)

[Advertising Packages](#)

Clarion Magazine has a number of advertising packages available for third party vendors and others interested in reaching Clarion developers.

(Dec 7, 1999)

[Product Review: ProDomus Translator Plus](#)

Software is a global business, and if your programs travel the world they'll need to know the local customs and language. That's when an internationalization/localization utility like ProDomus's Translator Plus becomes indispensable.

(Dec 14, 1999)

[The Novice's Corner: Understanding Clarion Code](#)

The Novice's Corner continues with declaring variables and utility program for deleting old Windows temporary files.

(Dec 14, 1999)

[Open Source Update: Date Fix & DDE Classes](#)

Additions to the Clarion Open Source Project include Phil Will's leap year date calculation fix for Clarion 5 and earlier and DDE classes (includes a class for accessing Clarion).

(Dec 14, 1999)

[ABC Embeds Are Easy](#)

Tom Giles sifts through the huge number of ABC embed points and comes up with the ones you're most likely to need.

(Dec 21, 1999)

[December 1999 News](#)

Clarion world news: product announcements, upcoming events, and more.

(Dec 21, 1999)

[Edit-In-Place CheckBoxes Done Right](#)

ABC has a lot of edit-in-place functionality, but EIP check boxes are a bit quirky. Pete Halsted shows how make your browse's checkboxes look and act they way they should.

(Dec 21, 1999)

[A Class Wrapper For Files](#)

Imagine you've been asked to write just one piece of cost estimating code that can be used on several quite different sets of files. Jim Kane shows how it's done.

(Dec 21, 1999)

[Clarion Magazine Holiday Schedule](#)

The Clarion Magazine office will be closed for the holidays from December 21, 1999 through January 2, 2000. The next issue will be published January 4, 2000. A Merry Christmas and Happy New Year to all!

(Dec 21, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resources

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.

\$6.²⁵/month[Main Page](#)[Log In](#)[Subscribe](#)[Frequently Asked Questions](#)[Site Index](#)[Article Index](#)[Author Index](#)[Links To](#)[Other Sites](#)[Downloads](#)[Open Source](#)[Project](#)[Issues in](#)[PDF Format](#)[Free Software](#)[Advertising](#)[Contact Us](#)

Press Release

December, 1999

Clarion Magazine's New Look

December 7, 1999

Many magazines, whether print or electronic, regularly undergo graphic redesign. Clarion Magazine has just been through one of these cycles, and although it's been a sometimes grueling process, we're happy with the results, and we hope you are too.

To get the full benefit of the design you'll need to use a browser that supports style sheets. Almost all of you do. IE 4.x and up and Netscape 4.5 and up are probably good choices. One way to check your browser is to look at the heading at the top of this page. If the words "Clarion Magazine's New Look" are in a dark blue, your browser supports basic style sheet functionality. The text on this page should also be in a sans-serif font (Verdana is our first choice, then Arial, then Helvetica). [Click here](#) for more about Verdana.

You'll also want to have your monitor set up to display more than 256 colors, and again this almost certainly the case.

In time we hope to convert all of the back issues to the new format, but this is a non-trivial task. For now, you'll notice a change in format when you look at most of the articles posted before December 1, 1999.

Aside from the changed graphics, this new format makes better use of available real estate. Ads on feature articles now appear on the left side of the page, leaving more space for article text. All articles now have a search button for easy access to the information contained in Clarion Magazine, and many of the general information pages now have a common sidebar menu.

This is the first major redesign of Clarion Magazine, and no doubt it won't be the last. We're always looking for ways to improve content and delivery. If you have any comments or suggestions, feel free to email me at editor@clarionmag.com, or better yet

[About Our New Look](#)

(Dec 7, 1999)

[Special Report: ConVic '99 Clarion Conference](#)

(Dec 7, 1999)

[Template Writing Made Easy](#)

(Dec 7, 1999)

[The SQL Answer Cowboy](#)

(Dec 7, 1999)

[The Clarion Advisor - Listbox Styles](#)

(Dec 7, 1999)

[The Cranky Programmer: Nits And Bits](#)

(Dec 7, 1999)

[ABC Design Series: View Manager Part 2](#)

(Dec 7, 1999)

[New ClarionMag Link Buttons](#)

(Dec 7, 1999)

[ClarionMag Newsgroup Reminder](#)

(Dec 7, 1999)

[Free Utilities: Class Browser And Compile Manager](#)

(Dec 7, 1999)

[Advertising Packages](#)

(Dec 7, 1999)

TopSpeed**Clarion 5**
by TopSpeedFREE Microsoft
Internet
Explorer

post a message on the [ClarionMag newsgroups](#).

Dave Harms

Copyright © 1999-2000 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.

published by
CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.²⁵/month**[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)**Feature Article**

December, 1999

A Nice Place For A Conference

by Simon Brewer

The words of Chris Livingstone, head of the Victorian CUG, seem so distant now. When first he mentioned another "ConVic" Clarion conference to me around a year ago, and suggested a fairly out-of-the-way location, I must admit I didn't think it would get off the ground. However, just over three weeks ago, Clarion faithful from around Australia ascended Mt. Buffalo, Victoria for [ConVic 99](#). For those who don't know, Mt. Buffalo is around 300 km (200 miles) north of Melbourne.

[click for full image](#)

This was the second ConVic conference – well, more of a retreat than a conference. A couple of years ago, I first discussed the idea of a regional developers' conference with the guys in my neighbouring state of Victoria. I was pleased to find that they were very keen on staging an event with a technical focus, but unlike traditional Devcons, they were keen to have a "live-in" conference where everyone could really get involved. And so ConVic 98 was born.

ConVic 98 was a great conference. Small by world standards, but brimming with ample Clarion technical detail and buoyed by the presence of Dave Harms, it was a great success. We immediately decided on another and planning began. ConVic 99 was going to be "bigger than Ben Hur" – or so we thought.

You could say we had our fair share of hiccups getting this conference going. We secured and lost not one but two international presenters and that cost us very dearly in time. Many things conspired to make promotion of the conference very low key and we

[About Our New Look](#)
(Dec 7, 1999)[Special Report: ConVic '99
Clarion Conference](#)
(Dec 7, 1999)[Template Writing Made
Easy](#)
(Dec 7, 1999)[The SQL Answer Cowboy](#)
(Dec 7, 1999)[The Clarion Advisor -
Listbox Styles](#)
(Dec 7, 1999)[The Cranky Programmer:
Nits And Bits](#)
(Dec 7, 1999)[ABC Design Series: View
Manager Part 2](#)
(Dec 7, 1999)[New ClarionMag Link
Buttons](#)
(Dec 7, 1999)[ClarionMag Newsgroup
Reminder](#)
(Dec 7, 1999)[Free Utilities: Class
Browser And Compile
Manager](#)
(Dec 7, 1999)[Advertising Packages](#)
(Dec 7, 1999)**TopSpeed****Clarion 5**
by TopSpeed

Developer PLUS

Your source for development tools and add-ons.

Your outlet for application sales.

Des Cousins

- Translator PLUS
- Date Tools
- Address Formats
- Lookups
- Finance

International Software

etc 2000

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

ended up with much less support than we were hoping for. The final figure was 34 attendees (plus several partners), or about 10% of Florida's attendance. In hindsight, I don't think that's too bad for this far-flung corner of the globe, and it was a better turnout per head of population than Florida!

At 1337 metres above sea level (that's around 4450 feet) in a historic wooden chalet, ConVic '99 kicked off mid-afternoon of Friday November 5th. Amongst those gathered were the national Clarion distributor and four of Australia's User Group leaders, one of whom is also a Team TopSpeed member.

The afternoon started well with the world's first public demonstration of Clarion 5.5 Beta by Ray Creighton. I threw in a collection of handy Clarion tips and then it was over to Des Cousins of Master Software to demonstrate his touch-screen Point-Of-Sale system. This system must be seen to be believed. Having taken up Clarion in early 1997, Des and his team have re-defined the term *User Interface*, the emphasis being fairly and squarely on the word "User." It's no wonder this remarkable system is winning supporters far and wide and Des was not backward in coming forward to tell his fellow Clarioneers how it all worked. Des proved to be somewhat more than an amateur magician during the conference, but there were definitely no smoke & mirrors in his applications!



[click for full image](#)

Fresh back from Florida Devcon, David Blundell and Andrew McPherson were able to fill us in on all the latest Clarion gossip and happenings to round the day off. No matter how much you read, talking to people who were there to soak up the vibe of the event and chat with "the names" can provide a different perspective. It was a great session.

After an easy night of socialising, Saturday did not start with a bang. The fog and rain set in so hard that you literally couldn't see out of the windows, and that's pretty much how it remained for the rest of the day. Fortunately, the action inside was just hotting up and we started off with a great session on multi-dll programming thanks to Bruce Cowan. I threw in some more tips and then it was over to Yogi Loechner.



[click for full image](#)

Yogi needs several paragraphs of his own. His shrink-wrapped package, Quicken QuickPayroll is probably the world's most successful Clarion-written package. Sold as an adjunct to Quicken's popular QuickBooks accounting suite, Yogi reckons he stopped counting "when sales got to 30,000 copies a couple of years ago." Simply put, there is seemingly nothing that Yogi cannot make Clarion do and his product is excellent. Complicated by the fact it must look and act exactly like the parent accounting system, it's a tribute to the incredible versatility of Clarion that it can be done – a real showpiece.

Controversially, Yogi is deliberately one generation of Clarion behind, so the product is actually written in Clarion 4 using legacy templates. That makes the smooth, fast, flicker-free results and the amazing level of features even more remarkable. Given that the product simply must not fail for any reason, his conservatism was understandable. He will soon be moving to Clarion 5 and ABC templates.

Before I leave Yogi, I must mention his end-user reporting tools. Move over TopSpeed

and other purveyors of end-user reporting tools: this is simply the Rolls Royce. When asked who'd buy a copy of it if it were available, around 70% of attendees nearly caused themselves an injury thrusting a hand skywards. Now we've just got to convince Yogi to sell it! In fact, and quite seriously, TopSpeed should look at it carefully to replace or enhance Report Writer. It's that good.

Rain washed out the planned outdoor activities but proved a blessing in disguise as it allowed us to slot in some extra impromptu sessions for the afternoon. I kicked off (not again!) with a session on converting Legacy applications to ABC bit by bit. The methodology I demonstrated was very well received, so if I can ever find time, I'll try and get something together for Clarion Magazine. This was followed by various demonstrations of software and rounded off by Des Cousens showing more behind-the-scenes details of his POS applications (still no smoke & mirrors!).



[click for full image](#)

Coming a close second to Australia winning the Rugby Union world cup, The Great ConVic Trivia Quiz was the highlight of the evening, with five teams vying for the major prize of a box of chocolates. Well, there were actually six teams, named for: Cowboy, Ferret, Barrington, Bayliss, Eggen and Rafalco, but Barrington symbolically stepped down before we got underway. A fantastic night was had by all, but I won't publish the final results lest they offend the personalities in question.



[click for full image](#)

On Sunday morning we were greeted with a clear sky and panoramic views of the plains way below us. While the 250 metre (850 foot) waterfall thundered away close by, we thundered into more great Clarion presentations. Owen Bruncker was first off the mark with an overview of the Internet, the history, the protocols, and how simple add-ons, such as Catalyst Socket Tools, could be utilised with Clarion to gain access. He demonstrated an NNTP-based application to read the TopSpeed newsgroups. His application contained an excellent abstraction on the

implementation of ActiveX controls, particularly Catalyst, and he shared this abstraction with everyone.

Yogi chimed in for another show-stopping demonstration, this time on template writing. I didn't mention above that Yogi is as entertaining as his demonstrations are interesting. Needless to say, the crowd was riveted. After other shorter demonstration sessions, David Blundell presented a look at some of the best third-party add-ons available for Clarion. David is definitely Australia's third-party guru – possibly the world's. There is seemingly no product he doesn't know about, and he can just about tell you what each third party developer's kids have for breakfast each morning! The background information was as good as the demonstrations (except for the kids' breakfast stuff!).

Next, Andrew McPherson presented a look at Windows NT Terminal Server Edition and Citrix Metaframe. Needless to say, these are products gaining a huge amount of momentum and are providing another handy string in the bow of the average Clarion developer.

To wrap up the conference, we had a plenary session entitled "The Declining Art of

Clarion Programming", the title being derived from a recent article in an Australian computing magazine. In the room were eight people (about a quarter of conference attendees) who had been using Clarion for less than 12 months which soon dispelled the notion of decline. We then looked to some of the success stories within, and newly-found ones (such as eData) which clearly wiped the idea. Coupled with the new directions of TopSpeed and the new products promised, we concluded that Clarion is definitely not in decline.



[click for full image](#)

That hoary old chestnut of Clarion promotion was again brought up and the general agreement was that the TopSpeed marketing gurus still haven't got it right. No suggestions as to how to get it right exactly, but there was general agreement that getting Clarion mentioned in more magazine articles or seeing some advertising would be great. It was also agreed that flaunting success stories (such as Yogi's) would also reap benefits.

Overall, this was a superb conference. The support from local Clarion developers was fantastic and we once again proved that you don't need a cast of thousands to have a really beneficial event. Careful planning and attention to detail ensured the event was also a fiscal success.

It'd be remiss of me if I didn't mention the support we had from various sponsors. [First Ecom](#) generously donated a Clarion Internet hosting service including credit card processing facilities for e-commerce for one year valued at well over \$2000. [Capesoft](#), [Niketouch Solutions](#) and [Clarion Magazine](#) also donated prizes to a total value in excess of \$1000. Attendees were offered various discount schemes for purchases from these companies and a few others, including the Australian Clarion distributor Aeronaut Industries. Our sincere thanks go to those companies for supporting us so generously.

That's our conference in a nutshell – actually, a very long-winded nutshell. No apologies for wasting all those extra electrons on detail, but I wanted to be concise for four good reasons:

1. to inspire a few more of you out there to get together and do something similar – there is no substitute for an event like this and the benefits are very real
2. to remind you that you don't need hundreds of people to hold an event like this
3. to put in an unashamed mention of Yogi's reporting tools, and,
4. to get you interested in the next ConVic (look out for announcements next year)

See you at a future ConVic!

[Simon Brewer](#) is a Senior Analyst with Email Major Appliances, Australia's largest manufacturer of whitegoods and major Clarion users. In his spare time he is also the President of the South Australian Clarion User Group and a co-organiser of the ConVic conferences.

[About Our New Look](#)

(Dec 7, 1999)

[Special Report: ConVic '99 Clarion Conference](#)

(Dec 7, 1999)

[Template Writing Made Easy](#)

(Dec 7, 1999)

[The SQL Answer Cowboy](#)

(Dec 7, 1999)

[The Clarion Advisor - Listbox Styles](#)

(Dec 7, 1999)

[The Cranky Programmer: Nits And Bits](#)

(Dec 7, 1999)

[ABC Design Series: View Manager Part 2](#)

(Dec 7, 1999)

[New ClarionMag Link Buttons](#)

(Dec 7, 1999)

[ClarionMag Newsgroup Reminder](#)

(Dec 7, 1999)

[Free Utilities: Class Browser And Compile Manager](#)

(Dec 7, 1999)

[Advertising Packages](#)

(Dec 7, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resourcespublished by
CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.²⁵/month**[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)

Feature Article

December, 1999

Simple Template Creation

By Don Childers

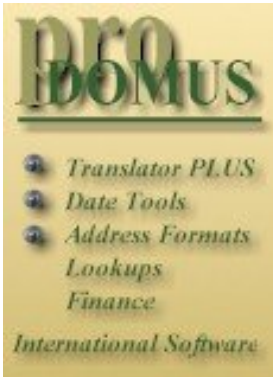
TopSpeed's Clarion is arguably the most productive software development system available today, and the language, the development environment and the Application Builder Classes contribute to this productivity. Yet many software development systems have productive languages, development environments and class libraries. In the area of productivity, Clarion's templates are the single feature that sets Clarion above those other systems.

With the advent of Wizatrns, which are essentially a template integration system, templates are poised to present developers with another tremendous increase in productivity. However, if templates and Wizatrns are to provide the maximum advantage, it's imperative that developers be able to create their own templates and integrate them into the Wizatron system.

Many Clarion programmers seem to believe that template writing is an advanced skill, reserved for those who know the Clarion language inside and out, eat Windows API calls for breakfast, and leap tall buildings in a single bound. Actually, any Clarion programmer who has successfully added a few lines of code to an embed point is ready for the challenge of creating a simple template. Template writing is one of the most effective ways to increase your abilities and your value as a Clarion programmer.

Template development should become a standard part of your development cycle, rather than a separate activity. Add template writing to the development process by taking a few minutes any time you write any hand code to decide if the code would be useful as a template. If the answer is yes, write the template immediately, while the purpose and function of the code is still fresh in your mind. Do this even if you're on a tight deadline. If you really don't have the time, or the task is beyond your current template-writing abilities, make a few notes in your simple version of the template so that you can go back and add that functionality later.

[About Our New Look](#)
(Dec 7, 1999)[Special Report: ConVic '99
Clarion Conference](#)
(Dec 7, 1999)[Template Writing Made
Easy](#)
(Dec 7, 1999)[The SQL Answer Cowboy](#)
(Dec 7, 1999)[The Clarion Advisor -
Listbox Styles](#)
(Dec 7, 1999)[The Cranky Programmer:
Nits And Bits](#)
(Dec 7, 1999)[ABC Design Series: View
Manager Part 2](#)
(Dec 7, 1999)[New ClarionMag Link
Buttons](#)
(Dec 7, 1999)[ClarionMag Newsgroup
Reminder](#)
(Dec 7, 1999)[Free Utilities: Class
Browser And Compile
Manager](#)
(Dec 7, 1999)[Advertising Packages](#)
(Dec 7, 1999)**TopSpeed****Clarion 5**
by TopSpeed



An Approach To Template Development

The process of writing, implementing and testing a template takes a bit more time than writing the equivalent hand code, so the best approach is to develop the functionality in procedure embed points. Then convert those embeds to a template after the code is proven. In a recent discussion on the TopSpeed.Products.C5EE newsgroup, someone wondered how to color the background of the currently selected field on a window. One of the responses advised declaring the local variable `LastField` as a `LONG`, then placing the following three lines of code in the `TakeSelected` event of `ThisWindow`, at priority 4000.

```
LastField{PROP:Background} = COLOR:White
SELECTED(){PROP:Background} = COLOR:Blue
LastField = SELECTED()
```

This code simply changes the color of the field represented by `LastField` to white, changes the color of the currently selected field to blue, then saves the field equate value of the currently selected field as `LastField`. I thought this could be a useful addition to my toolkit, so I decided to test the code in a current project and see if I liked the results. For my initial testing, I chose a simple procedure in a small application, to keep the overhead to a minimum. I also have a testbed application that I use for developing more complex templates.

Embedding the code as instructed worked, but the result wasn't quite as expected. Three-D windows use colors for buttons and checkbox text, but this code turned those controls white as you tabbed past them. Even worse, during the first execution of the code, `LastField` was equal to 0, which forced the background of the window itself to white!

The code needed to address these issues to be usable with 3D windows. First, it needed to leave the background color of the window alone. Second, it needed to store the color of the control it was about to change, and restore that color to the control when another control was selected. Finally, I decided to change the variable name `LastField`, so that it wouldn't be confused with the `LastField()` function. I chose `OldField` as the replacement name.

The background color of the window could be protected by an `IF` test. Storing the color of the control about to be changed required a new local variable, `OldColor`, to store the color code. `OldColor` must be a `LONG` to store a color value. As in the initial example, this code should be placed in the `TakeSelected` event of `ThisWindow`, at priority 4000.

```
IF OldField
  OldField{Prop:Background} = OldColor
END
OldColor = SELECTED(){Prop:Background}
OldField = SELECTED()
SELECTED(){Prop:Background} = COLOR:Blue
```

The `IF` test wrapped around the second line prevents the window color from being changed by skipping the background color assignment if `OldField = 0`, as it does when the first field on the window is selected. Zero is the control equate for the window itself.

Line four saves the color and line five saves the control equate of the currently selected field for later restoration. The sixth line colors the background of the currently selected field blue. These six lines of code will assure that all controls have their normal colors, except the currently selected control, which will have a background color of blue.

Problem solved. Move on to the next programming task, and just remember these six lines of code in case there's ever a need to color the background of the currently selected field on a window, right? WRONG! Take a little extra time, turn these six lines of code (and the two local variables... you'd forgotten them already, right?) into a template, and the next time you need the capability, you're only three mouse clicks from thought to implementation.

There are a number of template-writing tools available, including the Template Wizard, that can be used to shorten the template writing process. Learn to use one, and your efforts will be even more productive. For this example, writing the template by hand will be more instructive.

Templates are simply text files with a TPL or TPW extension. When you register a template the registry always asks for a TPL file. But having all your templates in one big TPL can be unwieldy, so you have the option of creating TPW files and linking them to the TPL with the #INCLUDE statement. Because the Clarion IDE is still 16-bit, you should follow the DOS 8.3 file naming convention. For this exercise, use your favorite text editor to create a file called MYTPL.TPL in your CLARION5\TEMPLATE directory. This file will become the glue that holds your collection of templates together, and you'll develop and store each template you create in a separate TPW file.

```
#!This file contains my collection of custom templates
#TEMPLATE(MyTpl, 'My Template Classes'), FAMILY('ABC')
#!SpotLite - Accent the currently selected field
#INCLUDE('SPOTLITE.TPW')
```

The first line of MYTPL.TPL is a template comment line (denoted by #!) that describes the purpose of the file. The second line uses the #TEMPLATE command to declare the template set. The third line specifies that the SPOTLITE.TPW file should be included in the processing of this template set. I decided to name the feature SpotLighting, because it provides a focus for the user, much as a spotlight helps the audience focus on the most important action on a stage.

After completing the MYTPL.TPL file, create the SPOTLITE.TPW file to contain the template itself.

Before writing the first line of any template you have to make a decision. What kind of template should this be? The simplest form of template, the #CODE template, embeds code in any embed point where it is used. This template needs to place code in specific locations whenever it's used, so a code template probably isn't the best choice.

#EXTENSION templates place code in embed points specified by the template writer, making an extension template the perfect fit for this task. #CONTROL templates are just extension templates that include one or more controls. Since there's no control associated with this task, a control template isn't needed. The fourth type of template, the #PROCEDURE template, is for creating complete procedures, rather than additions to existing procedures, so the initial choice of an extension template is correct. An extension template requires a template name and description. Once you've decided on those you can write the #EXTENSION statement.

```
#EXTENSION(SpotLite,'Accent the currently selected control')
```

A template's primary purpose is to generate source code during the code generation phase of development. To do this successfully, the template also needs to tell the developer what it can do, and ask the developer any questions it needs answered to perform the task. To determine the questions that should be asked of the developer during the design phase, review the code developed in the test procedure for options that could be left to the developer. The spotlight color is an option that should certainly be left to the developer. You can get this information using the #DISPLAY and #PROMPT statements.

```
#DISPLAY('This extension will set the color of the')
#DISPLAY('currently selected control to a developer-')
#DISPLAY('specified value and return the control to ')
#DISPLAY('the original color when a different control ')
#DISPLAY('is selected.')
#PROMPT('Selected Field Color',COLOR),%SelectedColor
```

The series of #DISPLAY statements tells the developer what the template is designed to accomplish. The #PROMPT command displays a prompt, accepts an input and stores the value selected for later use. In this version, the template displays 'Selected Field Color' as the prompt. The COLOR keyword used as an input parameter displays the standard Windows color dialog and allows the developer to select a color. The color selected by the developer is placed in the %SelectedColor variable.

Now that the developer is up to speed about the purpose of this template, and the template has collected the information it needs to do the job, it's time for the code generation part of the template. Code needs to be generated in two places. First, there are two local variables that need to be declared. After the variables are declared, the actual working code needs to be placed in the TakeSelected method of the WindowManager.

In the template language, you place code in a certain embed by wrapping the code in an #AT . . . #ENDAT phrase. The two local variables need to be placed in the Data Section of the procedure.

```
#AT(%DataSection)
OldField LONG !for use by SpotLite template
OldColor LONG !for use by SpotLite template
#ENDAT
```

The OldField and OldColor lines are the first lines of code in the template that don't start with a # character. Any line in a template that starts with a # is considered template code. If there is no leading #, the line is considered source code to be generated based on the surrounding template code. These four lines of code create the OldField and OldColor variables, both LONG data types, in the data section of the procedure.

Finally, the template needs to generate the actual field color control code in the TakeSelected method of the WindowManager. You can determine the #AT location for this block of code by examining the ABC templates, or by using the Template Wizard to generate a skeleton template from the test procedure. In a future article I'll use the template wizard to demonstrate ways to speed up template development.

```
#AT(%WindowManagerMethodCodeSection,'TakeSelected','←
      ( ),BYTE'),PRIORITY(4000)
  IF OldField
    OldField{Prop:Background} = OldColor
  END
  OldColor = SELECTED(){Prop:Background}
  OldField = SELECTED()
  SELECTED(){Prop:Background} = %SelectedColor
#ENDAT
```

The actual embedded code is essentially identical to the code in the test procedure. The only difference is in the line `SELECTED(){Prop:Background} = %SelectedColor`. This assigns the color selected by the developer as the background color of the currently selected field. The test procedure code assigned `COLOR:Blue` to the control.

When the template development process is complete, save and register the template, then comment out the hand code in the test procedure and add the template to the procedure. Generate, compile and test for proper final results, and you have a tool that will be in your toolkit many projects from now, instead of some code in an embed point somewhere that you don't know you could find if you really needed it again.

One handy feature of the template approach to software development manifests itself nicely in this example. Instead of placing this template on one procedure and having it apply to that procedure, you can also place it as an extension template in the Global section of an application, and it will be applied to every procedure in the application. You don't need to make any changes to the template! If you do this, spotlighting the current field in every procedure in your application is only three mouse clicks from thought to implementation.

Using the techniques in this article, plus some advanced study of the `#PROMPT` and `#AT` template commands, you can begin turning that 'write once and forget it' hand code into useful tools for future development projects.

Create and register the template, add it as a global extension to your favorite project, then take a look at a few of the browses and forms to see what an impact this simple tool can have on the appearance of any application.

[Click here](#) for the complete source code.

*[Don Childers](#) has been programming in Clarion since 1988. He was a co-author of *Tips, Tricks and Templates for CDD* and published a set of add-on templates for early versions of Clarion for Windows.*

[About Our New Look](#)

(Dec 7, 1999)

[Special Report: ConVic '99 Clarion Conference](#)

(Dec 7, 1999)

[Template Writing Made Easy](#)

(Dec 7, 1999)

[The SQL Answer Cowboy](#)

(Dec 7, 1999)

[The Clarion Advisor - Listbox Styles](#)

(Dec 7, 1999)

[The Cranky Programmer: Nits And Bits](#)

(Dec 7, 1999)

[ABC Design Series: View Manager Part 2](#)

(Dec 7, 1999)

[New ClarionMag Link Buttons](#)

(Dec 7, 1999)

[ClarionMag Newsgroup Reminder](#)

(Dec 7, 1999)

[Free Utilities: Class Browser And Compile Manager](#)

(Dec 7, 1999)

[Advertising Packages](#)

(Dec 7, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resourcespublished by
CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.25/month**[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)**The Clarion Advisor**

December, 1999

The SQL Answer Cowboy

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

If you have an SQL question for Andy, [click here](#).

Question:

We use your templates for some of our browses but in other places it just isn't practical (switching DCT properties from TPS to SQL etc where your browses wouldn't work with TPS). Lately we tried having Clarion generate a browse pointing at an SQL file and it worked fine, except the browse takes forever to switch between tabs (primary key is a single long and unique (takes 30 seconds), where others can take up to 160 seconds). Is there some simple PROP or switch we could use to speed these browses up even just a little? Your template browses, when browsing the same file, take less than a second switching from any sort order we have tried!

Ivan Faulkner

Answer:

Using PROP:SQLFILTER or PROP:ORDER you can reset the ORDER BY. Here is some testing you can do: Turn on the driver trace program (drvtrace.exe) so you can monitor the Sql Statements that Clarion is generating during the execution of the program. You will more than likely find that several fields are in the ORDER BY that you don't wish to have. Using the PROP:ORDER or the PROP:SQLFilter to modify the statement that is generated will alleviate some of your problems. Under certain circumstances, Clarion will

[About Our New Look](#)
(Dec 7, 1999)[Special Report: ConVic '99
Clarion Conference](#)
(Dec 7, 1999)[Template Writing Made
Easy](#)
(Dec 7, 1999)[The SQL Answer Cowboy](#)
(Dec 7, 1999)[The Clarion Advisor -
Listbox Styles](#)
(Dec 7, 1999)[The Cranky Programmer:
Nits And Bits](#)
(Dec 7, 1999)[ABC Design Series: View
Manager Part 2](#)
(Dec 7, 1999)[New ClarionMag Link
Buttons](#)
(Dec 7, 1999)[ClarionMag Newsgroup
Reminder](#)
(Dec 7, 1999)[Free Utilities: Class
Browser And Compile
Manager](#)
(Dec 7, 1999)[Advertising Packages](#)
(Dec 7, 1999)**TopSpeed****Clarion 5**
by TopSpeed

generate an order by clause that has every field in the primary key, and also every field in the browse. Hence your delay.

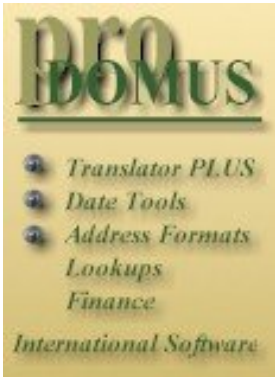
Copyright © 1999-2000 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca.



Developer
PLUS

Your source for
development tools
and add-ons.

Your outlet for
application sales.



PRO
DOMUS

- Translator PLUS
- Date Tools
- Address Formats
- Lookups
- Finance

International Software



etc
2000

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

published by
CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.25/month**[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)

Feature Article

December, 1999

A Question of Style

by Mike Pickus

Want to spruce up those old listboxes? Of course, you can color a column. With a little work, you can even conditionally color a cell. But what if you could customize each cell with its own picture and font? You know, give it a hint of style.

That's what the Style checkbox in the listbox formatter is for. You can define sets of display attributes as styles using the property syntax and then apply selected styles to individual cells.

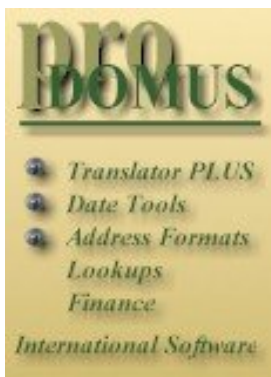
If you've tried the Style checkbox, you probably didn't like the results. The templates are incomplete in this area, and don't generate the code to use Style as they do for the Color checkbox. Checking the Color checkbox adds four LONGs to the browse queue for each queue field. The Style checkbox should add one LONG for each queue field. That said, here is a kludge to let you use Style with a template generated browse queue.

This is an example of a currency exchange listbox to display the country, the current and previous exchange rates, and the percentage change. See Listing 1 for the data file format. In the browse that displays this file you'll want to show the currency of the country, with a positive percentage change emphasized with a color and font change.

You'll need a local variable for the percentage and four Style LONGs, as shown in Listing 2.

Create a browse procedure. Select the listbox formatter by right clicking on the listbox. Add the country and rate fields from the file and the percentage and style fields from local data. Make sure the style fields appear at the end, *after* the file fields. Format as you would normally but set the width of the Style fields to 0. Check the Style checkbox on the Current rate, the Previous rate, and the Percentage. Save the window.

[About Our New Look](#)
(Dec 7, 1999)[Special Report: ConVic '99
Clarion Conference](#)
(Dec 7, 1999)[Template Writing Made
Easy](#)
(Dec 7, 1999)[The SQL Answer Cowboy](#)
(Dec 7, 1999)[The Clarion Advisor -
Listbox Styles](#)
(Dec 7, 1999)[The Cranky Programmer:
Nits And Bits](#)
(Dec 7, 1999)[ABC Design Series: View
Manager Part 2](#)
(Dec 7, 1999)[New ClarionMag Link
Buttons](#)
(Dec 7, 1999)[ClarionMag Newsgroup
Reminder](#)
(Dec 7, 1999)[Free Utilities: Class
Browser And Compile
Manager](#)
(Dec 7, 1999)[Advertising Packages](#)
(Dec 7, 1999)**TopSpeed****Clarion 5**
by TopSpeed



On the Procedure Main Properties window, press the '...' button to the right of the Window button. Edit the order of the #FIELDS so that S1 follows the current rate and S2 follows Previous. See Listing 3.

Listing 1

```
Style                FILE,DRIVER( 'TOPSPEED' ),PRE( STY ), ←
                    CREATE, BINDABLE, THREAD
PKey                 KEY( STY:UID ), NOCASE, OPT, PRIMARY
Record              RECORD, PRE( )
UID                 LONG
Country             STRING( 20 )
Current             DECIMAL( 11, 6 )
Previous            DECIMAL( 11, 6 )
                    END
                    END
```

Listing 2

```
! Local Data
Percentage          DECIMAL( 7, 3 )
S1                  LONG, AUTO ! Style
S2                  LONG, AUTO ! Style
S3                  LONG, AUTO ! Style
```

Listing 3

```
! Before
#FIELDS( STY:Country, STY:Current, STY:Previous, ←
         Percentage, S1, S2, S3 )
! After
#FIELDS( STY:Country, STY:Current, S1, STY:Previous, ←
         S2, Percentage, S3 )
```

All browse fields have a corresponding queue field, and the queue is created by the templates. By moving the #FIELDS in the list box formatter you're altering the order of the fields in the queue, without changing the appearance of the browse.

Now add code at two embed points. At the first, set the Style definitions after the window opens. This example uses the embed in `ThisWindow.Init`, Priority 8100. See Listing 4.

Listing 4

```
?List{PROPSTYLE:Picture,1} = '@N14`4~ F~' ! France
?List{PROPSTYLE:Picture,2} = '@N~L. ~14.2' ! Italy
?List{PROPSTYLE:Picture,3} = '@N~ £ ~14.4' ! United Kingdom
?List{PROPSTYLE:Picture,4} = '@N~kr ~14`4' ! Norway
?List{PROPSTYLE:Picture,5} = '@N~DM ~14`4' ! Germany
?List{PROPSTYLE:Picture,6} = '@N14_4~ mk~' ! Finland
?List{PROPSTYLE:Picture,7} = '@N14`4~ kr~' ! Sweden

?List{PROPSTYLE:FontStyle,8} = FONT:Regular
?List{PROPSTYLE:FontName,8} = 'MS Sans Serif'
?List{PROPSTYLE:FontSize,8} = 9
```

```
?List{PROPSTYLE:TextColor,8} = COLOR:Red
?List{PROPSTYLE:BackColor,8} = COLOR:White
?List{PROPSTYLE:TextSelected,8} = COLOR:White
?List{PROPSTYLE:BackSelected,8} = COLOR:Red

?List{PROPSTYLE:FontStyle,9} = FONT:Bold + FONT:Italic
?List{PROPSTYLE:FontName,9} = 'Courier New'
?List{PROPSTYLE:FontSize,9} = 10
?List{PROPSTYLE:TextColor,9} = COLOR:Blue
?List{PROPSTYLE:BackColor,9} = COLOR:White
?List{PROPSTYLE:TextSelected,9} = COLOR:White
?List{PROPSTYLE:BackSelected,9} = COLOR:Blue
```

(A nice addition to PROPSTYLE would be PROPSTYLE:Justification and PROPSTYLE:Indent.) At the second embed, set the listbox variables for each queue record at the BRW1.SetQueueRecord, Priority 2,500. See Listing 5.

Listing 5

```
! Set the currency picture
CASE STY:Country
  OF 'France';           S1 = 1
  OF 'Italy';           S1 = 2
  OF 'United Kingdom'; S1 = 3
  OF 'Norway';          S1 = 4
  OF 'Germany';         S1 = 5
  OF 'Finland';         S1 = 6
  OF 'Sweden';          S1 = 7
END
S2 = S1

! Do not divide by zero
Percentage = CHOOSE(STY:Previous = 0, 0, |
  (STY:Current - STY:Previous) / STY:Previous * 100)

! Set the percentage font -- blue (9) or red (8)
S3 = CHOOSE(Percentage > 0, 9, 8)
```

Figure 1 shows the browse in action.

Figure 1. The stylized list box.

Country	Current	Previous	% Change
Finland	5 6387 mk	5 6235 mk	0.270
France	6,3277 F	6,2211 F	1.714
Germany	DM 1,8866	DM 1,8549	1.709
Italy	L 1,868.00	L 1,866.00	0.107
Italy	L 1,868.45	L 1,876.37	-0.422
Norway	kr 7,8400	kr 7,7590	1.044
Sweden	8,2680 kr	8,5680 kr	-3.501
United Kingdom	£ 0.6179	£ 0.6114	1.063

Buttons: Insert, Change, Delete, Close

As you can see from the accompanying dictionary and application you can customize each cell with Style. And you have another tool in your tool chest.

Mike Pickus is a member of Team TopSpeed QA and hosts the VA/MD/DC TopSpeed User Group in McLean, Virginia. His latest project is a touchscreen POS application.

[About Our New Look](#)

(Dec 7, 1999)

[Special Report: ConVic '99 Clarion Conference](#)

(Dec 7, 1999)

[Template Writing Made Easy](#)

(Dec 7, 1999)

[The SQL Answer Cowboy](#)

(Dec 7, 1999)

[The Clarion Advisor - Listbox Styles](#)

(Dec 7, 1999)

[The Cranky Programmer: Nits And Bits](#)

(Dec 7, 1999)

[ABC Design Series: View Manager Part 2](#)

(Dec 7, 1999)

[New ClarionMag Link Buttons](#)

(Dec 7, 1999)

[ClarionMag Newsgroup Reminder](#)

(Dec 7, 1999)

[Free Utilities: Class Browser And Compile Manager](#)

(Dec 7, 1999)

[Advertising Packages](#)

(Dec 7, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resources

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.

\$6.²⁵/month[Main Page](#)[Log In](#)[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)[Article Index](#)[Author Index](#)[Links To
Other Sites](#)[Downloads](#)[Open Source](#)[Project](#)[Issues in](#)[PDF Format](#)[Free Software](#)[Advertising](#)[Contact Us](#)

Opinion

December, 1999

The Cranky Programmer Nits and Bits

Harrumph!!

So they thought they could keep the Cranky Programmer down by piling on so much work that he didn't have time to eat or sleep, did they?

Well, I sneer at their pitiful efforts (even though they DID work and I haven't eaten or slept in... slept in... ZZZZZZZZZZZZZZZZZ).

Oops, sorry. Now where was I...

Clarion 5.5 Beta 1 – the Big Teasy

In my meanderings through the newsgroups throughout the previous month, I had read quite a few enthusiastic endorsements of the new Clarion 5.5 Visual Property Editors, otherwise known as VPE. (If you are not familiar with the new features of C5.5, check out the recent [preview article](#) by Carl Barnes.)

After reading some messages that claimed vast productivity improvements because of them, I was definitely looking forward to trying them out myself.

Well, I don't know what version these people were using, but it couldn't have been the one I got. (Hmmm... on second thought, most of these raves were from Team TopSpeed members, and you can bet your booties that they have newer builds than the one on the CD. <sigh>)

In any case, I got my CD, installed C5.5 (let's see, that makes a nice round ten versions of Clarion currently installed on my machine), and loaded in an app. Actually, I loaded the one sample app that comes with 5.5; I wasn't about to use a real application for my first shot at a Beta 1 product.

[About Our New Look](#)

(Dec 7, 1999)

[Special Report: ConVic '99
Clarion Conference](#)

(Dec 7, 1999)

[Template Writing Made
Easy](#)

(Dec 7, 1999)

[The SQL Answer Cowboy](#)

(Dec 7, 1999)

[The Clarion Advisor -
Listbox Styles](#)

(Dec 7, 1999)

[The Cranky Programmer:
Nits And Bits](#)

(Dec 7, 1999)

[ABC Design Series: View
Manager Part 2](#)

(Dec 7, 1999)

[New ClarionMag Link
Buttons](#)

(Dec 7, 1999)

[ClarionMag Newsgroup
Reminder](#)

(Dec 7, 1999)

[Free Utilities: Class
Browser And Compile
Manager](#)

(Dec 7, 1999)

[Advertising Packages](#)

(Dec 7, 1999)





Zowie – look at that cool tree on the right. Scroll up and down through it and there are all the bits for the selected procedure: the list of files (oops – tables), local data, the window definition, filled embeds, extension templates used, the whole enchilada. Nice. Very nice!

I scrolled on down to the Window item, expanded the tree by clicking on the plus sign, double clicked on a window control and presto – there was the dialog box. Boy, I'm thinking, I could get used to this FAST. Then I saved the dialog and...

DOH!!!

The list collapsed and tossed the selection back to the very top of the list. A little more experimentation proved that no matter what I did, from where, the VPE tree collapsed and reset after every action. Ooh, what a timesaver *that* is. Kind of like having an ever-so-helpful assistant who puts the hammer away every time you reach for the next nail.

Argh! So tantalizing, so cool looking... and so incredibly annoying! Cruel TopSpeed! Bad TopSpeed to tease me like this!

Even more experimentation turned up that not only does the list reset, it returns to its default mode of collapsing the window, and expanding the tables, data, embeds and extensions items. Umm, guys, the idea is to preserve the tree the way *I* want it to be, where *I* want to be on it...

Guess the wait is on for Beta 2 to get a really functional VPE. (I still love the concept, though.)

As for other application tree enhancements, I really like the new "modified" tab. It shows the list of procedure in reverse modified order, i.e. from the most recently changed procedures at the top down to the neglected, never touched procedures at the bottom. Or maybe I should phrase that as: From the hinky* ones causing you all the problems down to the perfectly functioning procedures at the bottom. Your choice.

But I digress. Being the "never-say-enough!-because-then-they-will-stop-giving-you-cool-new-things" type that I am, it immediately made me want to see the date and time of the last modification as well. I kept scrolling to the right in vain, but nope – no soap. You listening, TopSpeed?

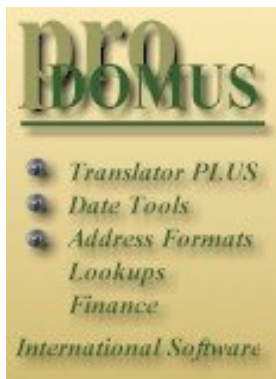
Can I Buy You A Map?

Is it just me, or has the Clarion IDE become progressively more stupid when it comes to dealing with compile errors in embed code? I've had to do some work in CW 2 lately, and it really brought home something that has been nagging at me for while.

In CW 2, ninety percent of the time you are taken to the correct embed point when there is a problem with the code you've written (ok, ok – the code *I've* written). Fix it and bang, you're on your way again. The only exceptions were for major errors where the compiler just had no CLUE as to what the code was meant to do.

In Clarion 5, though, well over half the time I get the ever popular "Edits made in generated source will be lost on next re-generate" message. It obligingly shows me the error, but not in a way that I can simply fix it and move along. Grrrrr.

Example: If I go to an EVENT:OpenWindow embed and type in a bogus procedure name (say, for example, "Bogus" - I'm SO imaginative, aren't I?) I will get the "Edits..."



message when I click on "Edit Errors" to fix the compile error.

Because of this, I find myself losing a fair amount of time, and often not even bothering with the "Edit Errors" button anymore. And THAT makes me cranky!

Late Flash! There's hope on the horizon. The same sequence as I used in the C5 example actually WORKS in C5.5 (it takes me to the "Bogus" procedure call in the embed point). Even better it take me there via the Embeditor so I can make related changes right then and there if necessary.

Ok. That does it. I WANT BETA 2! NOW!!

You Talking To Me?

What? You're *not* talking to me? Well, why not? Is everything so rosy in your Clarion world that you don't have anything to get even a TEENSY bit cranky about?

Didn't think so.

Drop me a line, get it off your chest. Ask me a question. Who knows, maybe I'll answer it in the column. Come on, here's your chance to get cranky and help the world at the same time.

I remain,
cranky@clarionmag.com

From the Cranky Programmer's Guide to Technical Terms, Volume 42, Section 3.50:

Hinky: Exhibiting somewhat flawed behavior; logically or functionally challenged; filled with unexpected features and surprises; i.e., someone else's fault, or written by your evil twin when you weren't looking.

[About Our New Look](#)

(Dec 7, 1999)

[Special Report: ConVic '99 Clarion Conference](#)

(Dec 7, 1999)

[Template Writing Made Easy](#)

(Dec 7, 1999)

[The SQL Answer Cowboy](#)

(Dec 7, 1999)

[The Clarion Advisor - Listbox Styles](#)

(Dec 7, 1999)

[The Cranky Programmer: Nits And Bits](#)

(Dec 7, 1999)

[ABC Design Series: View Manager Part 2](#)

(Dec 7, 1999)

[New ClarionMag Link Buttons](#)

(Dec 7, 1999)

[ClarionMag Newsgroup Reminder](#)

(Dec 7, 1999)

[Free Utilities: Class Browser And Compile Manager](#)

(Dec 7, 1999)

[Advertising Packages](#)

(Dec 7, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resourcespublished by
CoveComm Inc.

clarion magazine

Good help isn't that hard to find.

\$6.²⁵/month[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)

Feature Article

December, 1999

ABC Design: The ViewManager

or

Room With A View?

by David Bayliss

Part 2 of 2

In [part 1](#) of this article I discussed the ViewManager's initialization and cleanup methods. What's left is the code.

Application and Attitude

These three methods really embody the vast bulk of "the smarts" within the ViewManager. Their job is to construct a filter and order clause to provide a record set equal to the one currently requested. They also have an alternative agenda, which is to avoid resetting the VIEW's order and filter clause unless absolutely required. This avoids busy work in the view driver.

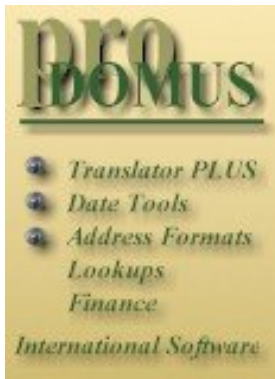
ApplyFilter PROCEDURE,VIRTUAL

This function really does three separate jobs. Firstly it constructs a filter equivalent of any range limits provided; then it concatenates any filters provided; finally, it applies the filter to the view and monitors the error condition.

The CASE statement is simply to branch between current, single and file range limits, all of which result in a filter of the form "field1 = xxx AND field2 = yyy" and the pair range limit which produces "field1 >= xxx and field1 <= yyy".

For the common case the code sits in a loop for each element of the range limit queue. For each element the field name is extracted from the file manager, then CasedValue is called to compute the right hand side of the equals sign for the given condition.

[About Our New Look](#)
(Dec 7, 1999)[Special Report: ConVic '99
Clarion Conference](#)
(Dec 7, 1999)[Template Writing Made
Easy](#)
(Dec 7, 1999)[The SQL Answer Cowboy](#)
(Dec 7, 1999)[The Clarion Advisor -
Listbox Styles](#)
(Dec 7, 1999)[The Cranky Programmer:
Nits And Bits](#)
(Dec 7, 1999)[ABC Design Series: View
Manager Part 2](#)
(Dec 7, 1999)[New ClarionMag Link
Buttons](#)
(Dec 7, 1999)[ClarionMag Newsgroup
Reminder](#)
(Dec 7, 1999)[Free Utilities: Class
Browser And Compile
Manager](#)
(Dec 7, 1999)[Advertising Packages](#)
(Dec 7, 1999)



CasedValue allows for the right hand side being a string (in which case quotes are used), the key being case insensitive (in which case an UPPER will be present of the field name and should be present on the constant name) and even the value containing quotes!

The range filter assignment line has an interesting tweak, which is the use of an inline choose statement:

```
CHOOSE ( I = 1, ' ', ' AND ' )
```

This solves the age old problem of having a list of items that you want separated (with an AND in this case). The traditional solution (using an IF statement before the concatenation) can make a simple loop look complex. The CHOOSE handles things in a very compact way.

The Pair code is a little more complex and illustrates nicely the treatment of key components which come before the component being range limited. If you look at the case within the loop the second branch is only taken for those major components. They are simply "=" limited, exactly the same as for a current value range limit. In other words, the standard ABC library deals with multi-component range limits *without any clever tricks being needed*.

The RRL-1th element is the lower bound of the range leaving the upper bound to be computed outside the loop.

The filters supplied by SetFilter are then appended in turn (there can be any number of them, each with a different ID). Each supplied filter is placed inside parenthesis to avoid any unexpected operation precedence problems.

Finally the filter is assigned and the error trapped.

ApplyOrder PROCEDURE,VIRTUAL

This method doesn't have any complexity. It just assigns the order clause and traps any errors.

ApplyRange PROCEDURE,VIRTUAL,BYTE,PROC

ApplyRange is where the system gets some attitude. The idea here is that sometimes the window manager knows "things have changed" and wants to alert the ViewManager that it needs to refresh itself. But the ViewManager shouldn't refresh itself if nothing of interest has actually changed. So ABC has the ApplyRange method.

For every range limit (other than current value) a mirror value is stored to reflect the state of the range limit the last time the browse was refreshed. When ApplyRange is called it checks the new values against those in the mirrors. If nothing has changed then ApplyRange simply returns; otherwise ApplyFilter is called to handle the changes in the data.

For the Pair case both the upper and lower bound have to be checked. This is done by comparing the right (which acts as a buffer) with the left. If there is a difference then left is assigned to right (for both bounds) and ApplyFilter is called.

There is a neat trick here: the Single clause is introduced by OROF not OF. This means that the Pair case will *also* fall down into this code.

The `File` case looks a bit more complex but it isn't. For the file range limit both left and right values are used (the left is the file being limited, the right is the file doing the limiting) so if the file doing the limiting has changed the new value is assigned to the buffer part of the `BufferedPairsClass` before the `ApplyFilter` is done.

Lights, Cameras ... Action

The following methods are really the ones that are called to *do things* to the view during normal operation of the `ViewManager`. They have names and semantics similar to equivalents in the underlying file managers and relation managers.

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

The purpose of this method is to allow a record to be cleared/prepared for sending to an Update (or similar code) for insertion. This work is done in three stages,

1. Each file buffer connected to the view is cleared. This is done by quizzing the view driver for what the files are and then simply calling `Clear` for each record.
2. Next comes a bit of intelligent. Suppose you have a browse that is range-limited to only display addresses in Florida. Then assume an insert button is pressed: it is reasonable to assume the new blank record should comply with the Florida range limit. So the range limit elements are stepped through and the key components of the sort order are filled in with the corresponding limiting value.
3. Finally any remaining "blank bits" from the primary record are filled in (such as any auto-increment key components). This is done by a call down into the `PrimeRecord` method of the underlying file manager.

Close PROCEDURE,VIRTUAL

Open PROCEDURE,VIRTUAL

These methods close and open the view respectively. They are extremely simple. The `Opened` flag allow them to be called on a "check if it is open" and "check if it is shut" basis without generating errors. The `Open` method also applies the presently active filters and order so that the static filter/order of the view (usually declared in the generated CLW file) doesn't have a chance to load any records before they dynamic filter/order (which override the static one) is applied.

SetSort PROCEDURE(BYTE OrderNumber),BYTE,VIRTUAL,PROC

This method switches the presently active sort order. After this call further calls to (for example) `SetFilter/Reset` mean "perform this action on the present sort order." This is one of the main benefits of this manager, as I detailed earlier.

The value returned means "did anything happen." This is part of the attitude mechanism: if a zero is returned the caller knows it doesn't have to progress with any code that would be required for a new sort order. For example, the browse watches this to see if a reload might be required.

Reset PROCEDURE(BYTE Locate),VIRTUAL

Reset PROCEDURE,VIRTUAL

`Reset` on a `ViewManager` (and on any ABC object for that matter) means "bring this object up to speed with any external data it references." In this context it means bring the view up to date with any data on the disk and position it relative to data in the underlying

file record buffers. The `Locate` byte denotes how many leading components of the presently active order are taken into account when performing the locate. In other words, if `Locate` is zero (or you call the unparameterised form of `Reset`) no location is done and the view is positioned at the beginning of the record set. If `locate` is (say) 2 then the first two fields from the order clause are used to perform the position.

Next PROCEDURE,VIRTUAL,BYTE Previous PROCEDURE,VIRTUAL,BYTE

`Next` and `Previous` read the next and previous records from the current data set. This is pretty much just a call to the underlying view. Additionally the error conditions are converted into ABC error levels.

If the record is valid according to the view criteria then the `ValidateRecord` method is called. This can specify one of `Ok`, `Filtered` and `OutOfRange`. The `OK` from `ValidateRecord` simply causes a return from the `Next/Previous` method with a `Level:Benign` error rating. An `OutOfRange` return is converted into a `Level:Notify`, which is interpreted by the rest of ABC as an "End Of File." In other words it stops all view processing until the next reset. The `Filtered` case is handled by simple continuation. This causes the outer loop to cycle causing the next record to be read from the view. In this way no "invalid" records will ever be returned from the `Next/Previous` methods.

ValidateRecord PROCEDURE,BYTE,VIRTUAL

This method doesn't really belong in this section but it is so closely related to `Next/Previous` that I decided to cheat. The default implementation of `ValidateRecord` is useless - it just returns `Record:Ok`. It is here so that people can override it (by dropping embed code into the `ValidateRecord` embed point) and thus define an additional programmatic filter for what does and doesn't constitute a valid record. `ValidateRecord` should return `Record:Ok`, `Record:Filtered` or `Record:OutOfRange` as defined above.

HouseKeeping

The remaining methods are really housekeeping and are designed to allow people to fiddle with the internal state of the `ViewManager` object without having to know its structure. As such they are generally simple to use and implement. They all require the previous use of `SetSort` which sets the sort order to be acted upon.

GetFreeElementPosition PROCEDURE,BYTE,PROTECTED,VIRTUAL

This returns the position within the present sort order of the field which is the free element. You can think of this as the "number you need to pass to `Reset` to get a search using fields up to (and including) the free element." For example, if you have a three component key, and have range-limited the first element, this function will return 2. Now if you have values in component one (the fixed element) and component two (the floating one) a `Reset (2)` will locate to the first record matching both of those elements (but ignoring the third).

GetFreeElementName PROCEDURE,STRING,VIRTUAL

This method returns the *bind name* of the free element, which allows derived classes to perform some kind of location/free element manipulation other than a simple reset. For

example the filtered locator class uses this function to construct a filter of the form
`SUB(GetFreeElementName(),1,2) = 'FR'` or similar resulting in
`SUB(CUS:FirstName,1,2) = 'FR'`.

SetOrder PROCEDURE(String Order),VIRTUAL

`SetOrder` is directly equivalent to setting the order clause of the present sort order for the underlying `ViewManager`.

SetFilter PROCEDURE(String Filter),VIRTUAL

SetFilter PROCEDURE(String Filter,String Id),VIRTUAL

These two methods manage the filtering system. What we were aiming for was to enable people to write additions to (for example) a browse that enabled additional filterings. The classic is a QBE filter or a multi-component locator. The traditional problem has been how to set a new filter without clobbering the one provided by somebody else. Using the one parameter `SetFilter` you can't; whatever you set overrides the filter provided by the filter prompt in the template. However the two parameter filter allows you to specify a unique identifier for the filter being passed in. Provided that identifier does not clash with anyone else's (be inventive, this *is* a string) then all the set filters will be concatenated before being sent to the view.

There is an implicit priority mechanism in that the strings are alpha-sorted before going to the view driver. This may be important efficiency-wise as many expression evaluators (including ours) perform left-to-right short-circuit evaluation of boolean conditions. (Put another way, if your doing something slow, put it last filter as the system *probably* will have thrown a record away before the code gets called).

Summary

Phew! We have finally gotten to the end of `ABFILE.CLW`. Kinda tiring, huh? Well yes, although it should be remembered that `ABFILE` is *by far* the largest module in the ABC system (45% bigger than `ABBROWSE`) and the relation manager is probably the most complex part of the system.

I trust that the tour has provided you some insight into the operation of "the spine" and also into the coding techniques, styles and methodologies employed.

That said, the number one lesson I *hope* you have gleaned is that all the code is there, it is all fairly approachable and once you understand it it really does begin to make sense.

Honest.

[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.

[About Our New Look](#)

(Dec 7, 1999)

[Special Report: ConVic '99 Clarion Conference](#)

(Dec 7, 1999)

[Template Writing Made Easy](#)

(Dec 7, 1999)

[The SQL Answer Cowboy](#)

(Dec 7, 1999)

[The Clarion Advisor - Listbox Styles](#)

(Dec 7, 1999)

[The Cranky Programmer: Nits And Bits](#)

(Dec 7, 1999)

[ABC Design Series: View Manager Part 2](#)

(Dec 7, 1999)

[New ClarionMag Link Buttons](#)

(Dec 7, 1999)

[ClarionMag Newsgroup Reminder](#)

(Dec 7, 1999)

[Free Utilities: Class Browser And Compile Manager](#)

(Dec 7, 1999)

[Advertising Packages](#)

(Dec 7, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resources

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.

\$6.²⁵/month[Main Page](#)[Log In](#)[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)[Article Index](#)[Author Index](#)[Links To](#)[Other Sites](#)[Downloads](#)[Open Source](#)[Project](#)[Issues in](#)[PDF Format](#)[Free Software](#)[Advertising](#)[Contact Us](#)

Review

December, 1999

ProDomus Translator Plus

**Reviewed by Tom Hebenstreit,
Reviews Editor**

Translator Plus is the newest offering in a long line of internationalization products from longtime Clarion third party vendor ProDomus.

And what is internationalization? It's the process of turning your programs into chameleons, i.e. giving them the ability to transparently change from one language to another without requiring any further programming on your part (once you have done the initial work, of course). In other words, internationalization means translating the interface of a program, and a tool like ProDomus Translator Plus is a means to achieving that end.

At its core translation is a process of substituting one thing for another *while the program is running*. Strings, prompts, icons, images, messages – basically every visual aspect of the application that presents information is a candidate to be swapped. The translator accomplishes this by taking the original item and looking it up in a file (or queue loaded from a file). If the original is found, it is swapped out for the translated value.

To use a simple example, the translator might see that you have a button with the text "Help" on your window. Before the window is actually displayed, the translator functions would look to see if there is any text that should be substituted for "Help". If it finds "Help" in its translation list, it will take the replacement value and use it instead. If you were translating from English to German, it might substitute and display "Hilfe" on the button.

As you can imagine, the translation file records contain, at a minimum, a field for the original string and another field for translated version of the string. In reality, a translator like PD Translator Plus uses other fields as well which help manage hot keys, categorize the types of items to be translated and more. Additionally, the translator should dynamically deal with picture tokens so that your program matches the regional display settings for Windows itself.

Major Features

[Product Review:
ProDomus Translator Plus](#)
(Dec 14, 1999)[The Novice's Corner:
Understanding Clarion
Code](#)
(Dec 14, 1999)[Open Source Update:
Date Fix & DDE Classes](#)
(Dec 14, 1999)



The first thing to understand about Translator Plus is that it is not a single, all-in-one product. It is a set of interrelated components consisting of a number of ABC compliant classes, templates and wrappers, coupled with a set of utilities that help to create, manage and maintain the files used for translation. Each of the components can be purchased and used separately, with the exception of a few classes that depend on the functionality of another class (as noted below).

Also you should be aware that Translator Plus requires Clarion 5 or higher, and only supports the ABC templates. If you need legacy support, you should take a look at the original PD Translator (not the Plus version reviewed here).

Here is a rundown of the available Translator Plus components and utilities:

Classes

Picture Translation Class: This class uses Windows Registry information to automatically translate Clarion picture tokens such as currency, number, date and time based on a specified locale identifier. What's a locale? In a nutshell, it is a collection of pre-defined settings that Windows uses to format the way that numbers, dates, times and currencies are displayed. To see some of the standard Windows locale settings take a look at the Regional Settings Properties dialog in your own Windows Control Panel.

The Picture Translation class includes more than 90 locales, and supports multiple locales at the same time. Another neat feature is that the currency locale can be different from the general locale/language. Where is that useful? Consider French, for example, where you could have currencies from Belgium, Luxembourg, Canada and (of course) France, all within the same language settings.

String Property Translation Class: This class allows any property to be changed (strings, icons, images, list headers and pictures, etc.) and consolidates the source and replacement properties into a single database file. The class loads strings when your program starts, and in a neat twist, will also export any strings that weren't found during program execution when the app closes. You can then update your database to add the missing translations.

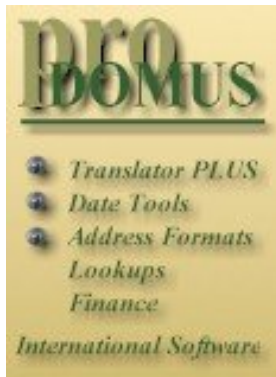
Options are provided for using either default or user defined files, drivers, variable file names and owner attributes.

This class also provides substitute procedures which let you redirect calls to Message, Stop and Halt to procedures of the same name which include translation. It seems to me that this can be a major timesaver if you are adding translation to an existing app which makes liberal use of the built-in Message() function, as those calls will be automatically re-routed through the new translation-aware version. Further, you can write new code without having to adjust your mindset.

Also included is a string merge function to merge variables with text.

Environment Class: This class uses Windows API calls and the String Property Translation class to set the Clarion Environment variables for your app. These include items like Collation, Case, Message Buttons, Digraphs, AP/PM text, full and abbreviated Month Names, Character Set, and Error Messages. A method for setting and restoring the text of any individual button is also provided.

By the way, you can find more information on the Clarion Environment settings by looking in the Clarion help under "Environment Files". Note that this class requires the



String Property Translation Class in order to function.

Character Class: This class gives you better support for extended ANSI characters, using Windows API functions instead of the more limited standard Clarion functions. The methods include such functions as `AsciiToAnsi()`, `Upper()`, `IsAlphaNumeric()`, `IsAllLower` and so forth. A simple `Proper()` function is also provided which capitalizes the first letter of a word.

Type Class: The Type class organizes properties by control and property type. To quote ProDomus: "Like the String class, this methodology is easy to implement. Unlike the String Class, the translation can reflect the context of the controls, providing the opportunity to better differentiate language contexts and syntax. It also makes it possible to edit groups of control types in consistent ways, i.e. capitalizing the first letter of all tool tips or using proper case for all window captions, menu items, and prompts."

Note that this class also requires the String Class in order to work.

A number of the classes can also work together with other classes when accessed via provided wrapper classes. For example, there is a String wrapper, a String-Picture wrapper, a Type-Picture wrapper and others.

Utilities

Where the templates and classes do the work while your programs are running, the following utilities are designed to help you during the process of creating and maintaining your translation files.

Source Manager Extraction Utility: This utility extracts translation strings from all types of files, converts various language file types and translates TRN files. It accommodates multiple projects and multi-DLL projects. You can also use it to batch process source files, automatically putting all text into a translation file.

Template Text Extraction Template: This tool exports template-generated window and report strings to a text file. It also has option to organize the exported strings by module, procedure, control type and more.

Environment Utility: If you work with 16-bit applications, you can use the Environment Utility to create "NLS" (National Language Support) and Clarion International Environment files for any locale installed on the your machine. Those can then be used to translate for your 16-bit apps.

Translation Assistant: This application is where you can update and organize the strings in your translation files. You can also create, edit, and import/export translated strings from translation dictionaries.

Extras

Translator Plus also includes an extra global extension template called PD Remove Class Include Files. This template can be used on a DLL to prevent unwanted classes from being compiled.

Finally, all of the classes include source code with the exception of the Picture Translation class. ProDomus has also stated that they intend to keep expanding the set of components as time goes on.

Installation

The Translator Plus install was a model example of everything a good install should be. It explained what it was going to do, automatically located where my Clarion 5 was installed, registered the templates, added Translator Plus to my C5 Accessories menu and basically proceeded very logically through all the many options and screens.

For example, I really liked that the opening screen of the install provided an excellent overview of both the installation and how to get started. Sections included Welcome, Installation, Files Installed, Passwords, Error Messages, Utility Registration, and Getting Started.

The full install took up close to 4 megs, but that included 1.7 megs worth of example files.

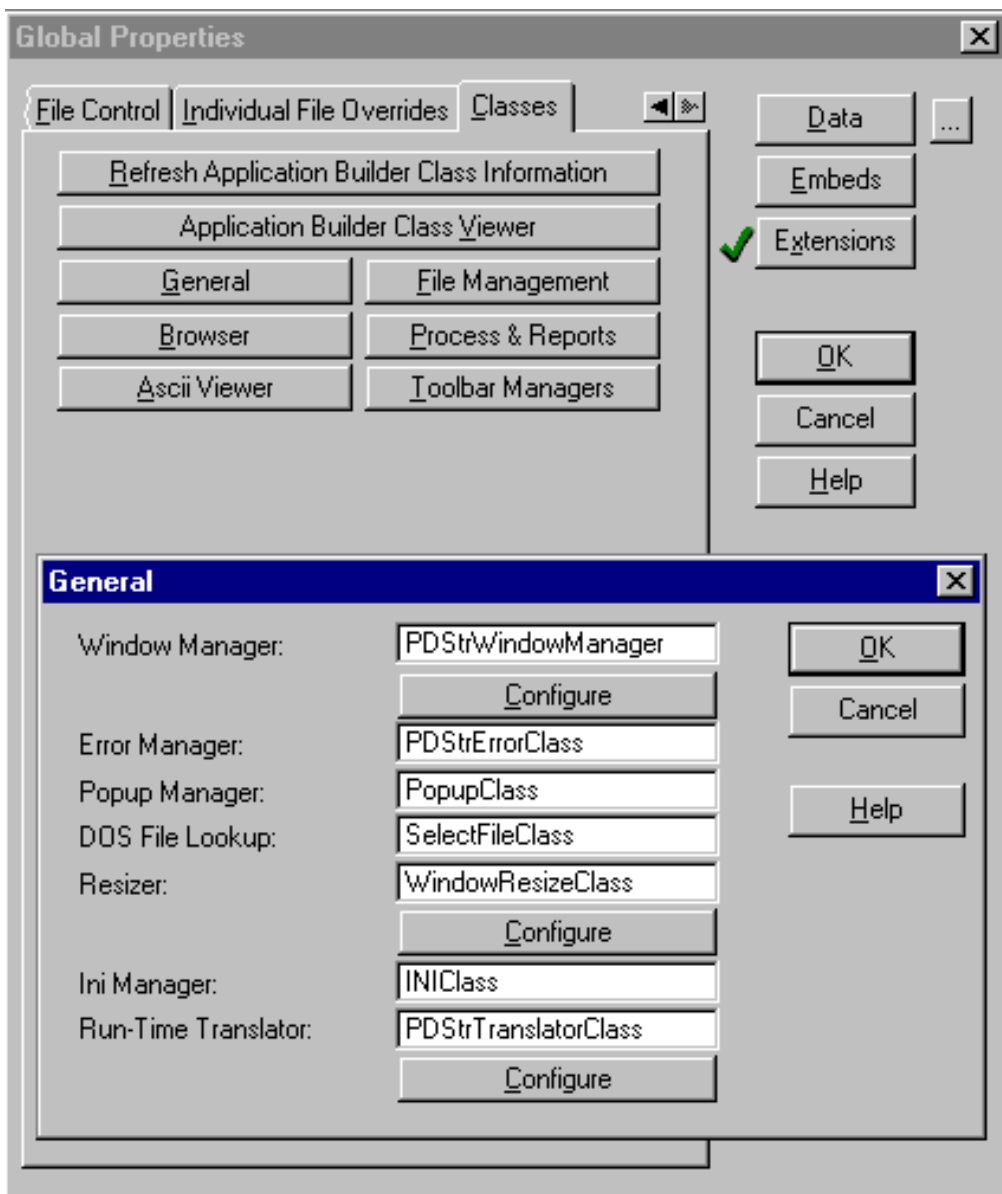
The biggest drawback to the installation process was that, as discussed above, Translator Plus contains nine separate components. Each component is password protected, and thus a full install such as I was doing required entering nine separate passwords. It wasn't hard, but involved typing a lot of gibberish *very* carefully. As a side note, each of the utility programs also requires registration when you first run it. I certainly can't fault ProDomus for wanting to protect their work, though.

All in all, the Translator Plus installation rates as excellent.

Implementation

The basic implementation of Translator Plus is about as easy as it gets: you add a global extension to your app, check the global Enable Run-time Translation box and specify that you want to use the Translator Plus classes in place of various default ABC classes (the Translator Plus classes are derived from the standard ABC classes, so you don't lose anything).

Figure 1. Specifying Translator Plus classes to be used instead of the defaults.



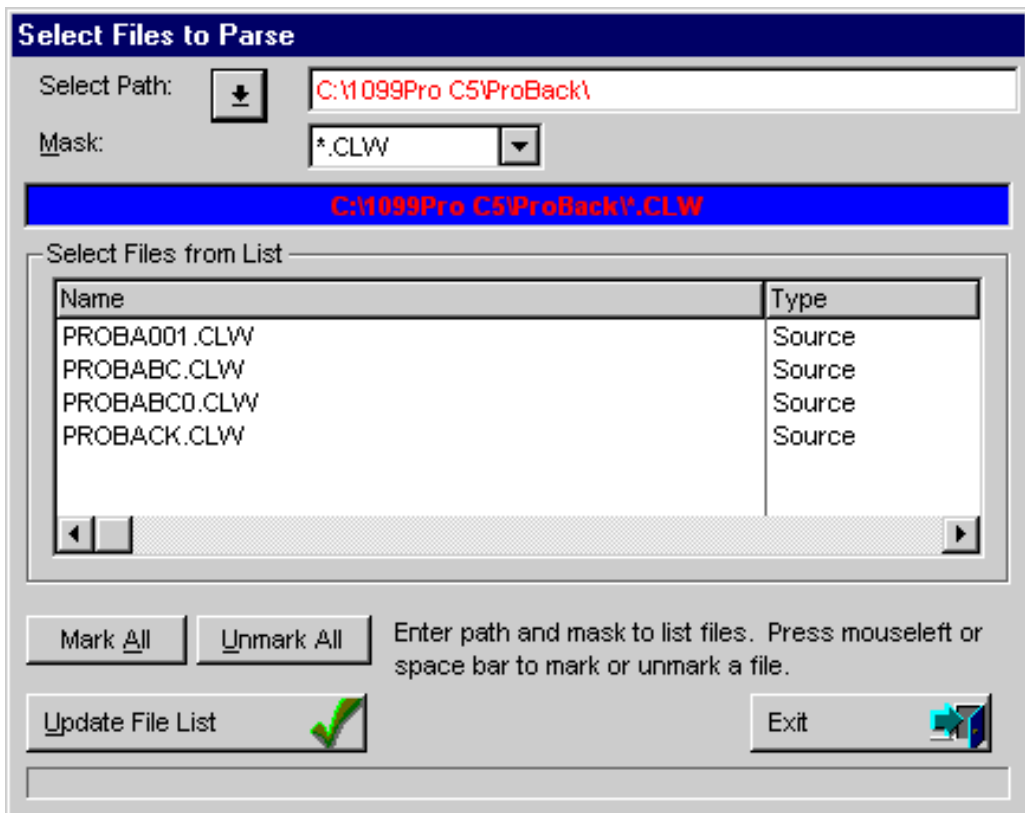
Now the real work begins, though.

Think of all the text in even a small application. We are talking strings, prompts, buttons, menus and, lest we forget, little details like tool tips and message strings. And don't forget the Clarion internal ABC messages generated by the FileManager and so forth either. *Every* bit of that text should have a translated counterpart. After all, the only thing worse than a cryptic error message is a cryptic error message in the wrong language!

This is where the Translator Plus utilities really come into their own. I am admittedly not an expert on translating applications, but after going through the process, I can just imagine how tedious it would be without these tools.

First off, I used the Source Extraction to grab all the basic strings from a small application. All that was required was to create a project (i.e., give the collection of files I wanted to translate a name) and then select the files from which text should be extracted.

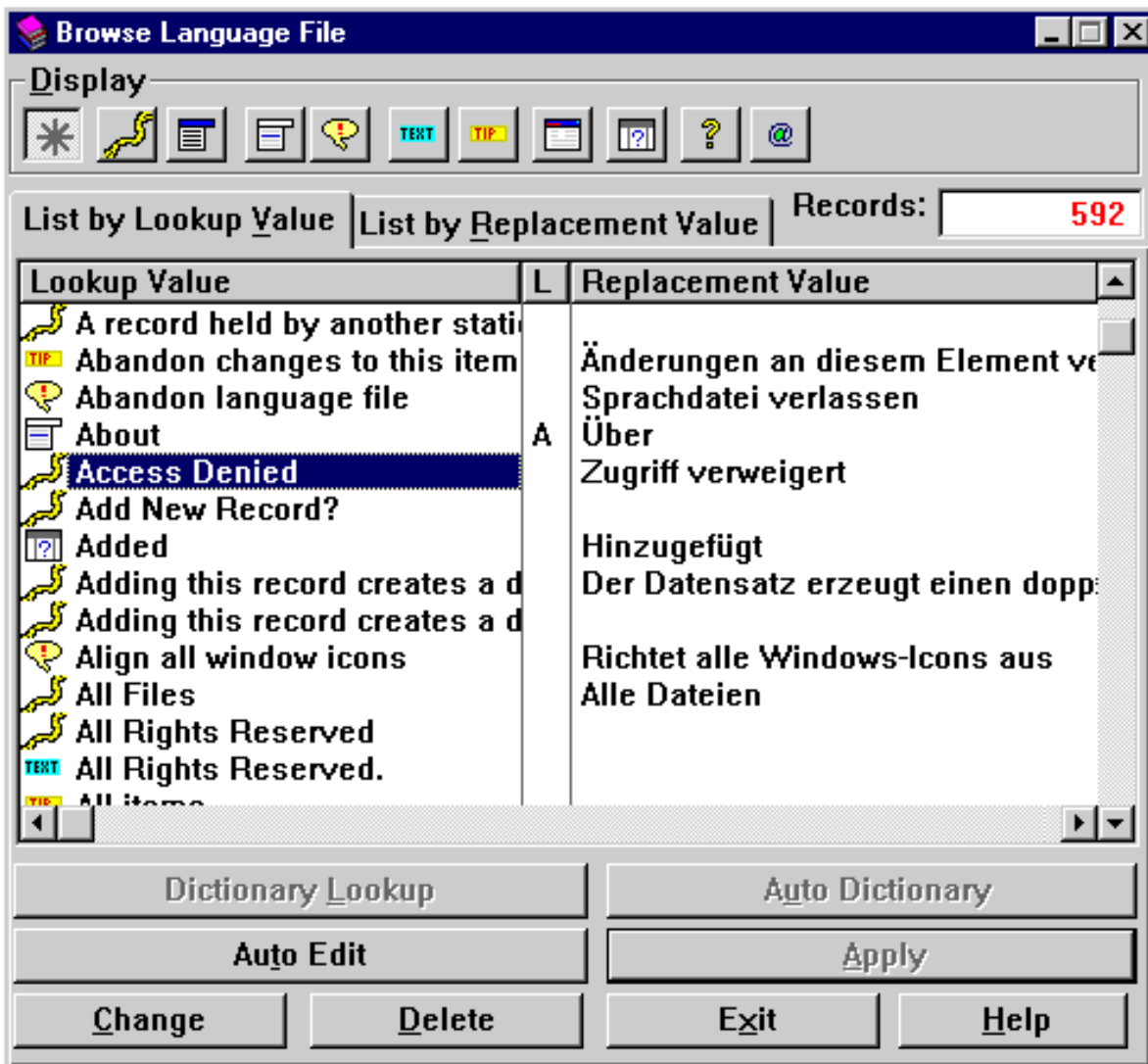
Figure 2: Selecting files for parsing by the Source Extraction Utility



Once I had the file, I used the Translation Assistant to go down the list and enter a few translations. I must say it is amazing how many strings even a small program contains. One of the programs I was experimenting with was a wizard utility that only has two procedures (one of them is just a function), and it produced over sixty translatable items. And this was before I had even run the utility to extract all of the standard Template generated messages and merge them into my translation file. Yikes!

Since my attempts at translation were, to put it politely, rather feeble (I don't remember much Junior High German, and English to English seemed rather pointless), here is an example of the Translation Assistant using the German translation file from the PD Translator Demo.

Figure 3: Translation Assistant with a German language translation



One thing I really did find myself wishing for in Translation Assistant was a string search feature so that I could quickly find items containing a particular bit of text.

Here are some snips of a form displaying (mostly) translated prompts and data. Note that all of these images are displaying the same exact data, just with different pictures based on the locale.

Figure 4: The base information in English

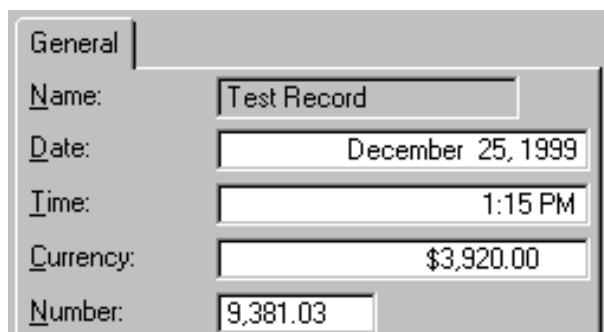


Figure 5: The translated version in German

Générale	
Nom:	Test Record
Date:	25 déc 1999
Heure:	13:15
Currency:	3.920,00 F
Number:	9.381,03

Figure 6: The translated version in French

Allgemein	
Name:	Test Record
Date:	25 Dez 1999
Zeit:	13:15
Currency:	3920,00 DM
Number:	9.381,03

In case you are wondering, if the translator can't find a translated equivalent for something it will display the item as originally written/designed. Your app won't pop up with blank screens or prompts, etc.

I haven't even touched on using the provided classes and wrapper in embed code (this review is long enough already), but I will say that Translator Plus does a good job of a) minimizing the changes you would need to make and b) documenting the class methods and properties for when you *do* need to use them.

Caveats

In the course of my correspondence with ProDomus, I asked Phil Will (the author of PD Translator Plus) a question about what PD Translator Plus *doesn't* handle. In other words, what are the areas to watch out for when translating an application?

He mentioned two specific areas: Third party tools and file data.

For third party tools, you would want ones that are "translation friendly," i.e. provide source code or another method to hook in the translator. Obviously, if you need to modify the templates to make them translatable, this requires a higher than average level of knowledge and tends to restrict upgrade options. Another thing Phil mentioned involved possible programming logic embodied within a template. For example, he noted one example that builds a very complex sentence using conditional concatenation - a real translation "no-no" according to Phil (how would you translate it?).

As far as file data goes, he gave the example of lookup files. You would need to accommodate switching data files, etc., to deal with language/locale changes.

It seems then, that for internationalization you really need to give some advance thought to *every* aspect of not only your application, but also the tools you are planning to build it with.

Performance

I noticed a performance hit on the first program startup for translated programs, as that is when the program loads all of the translation strings into memory. Obviously, this hit would vary depending on the size of the program, machine speed, and so on. For my own programs, I would probably want a nice splash screen that let the user know what was going on. Other than that, I didn't notice any performance degradation during actual program operation.

Running the test programs did reveal a few design issues that should be kept in mind if you plan to internationalize your own programs. Primarily they deal with screen spacing and layout issues. For example, you need to keep mind the maximum space that could be needed by prompts in *any* of your target languages. It will do you no good to design a gorgeous looking screen in English, just to see it fall to pieces when compact English prompts are translated into, say, longer German or French equivalents. The same goes for button text and anything else where you rely on sizing or spacing.

Documentation

The documentation I received was one of those "good points/bad points" situations.

A Windows help file is provided that does a good job of starting at the beginning and walking through the basics of the templates and classes. Good use is made of the ">>" and "<<" (Next/Previous) help link button so that all I had to do was just keep clicking on "Next" to move to the next logical topic.

An Adobe Acrobat PDF is also provided so that you have a printable manual (this is good). The drawback is that the file seems to be directly derived from the help file source document, and it has a lot of extraneous text that is obviously used by the Help compiler to create links, etc.

The sample programs provided with the product are instructive, but they don't *do* anything (which was a bit disappointing). By that I mean that they demonstrate how to implement the translation, but their translation files are empty. Thus, with the exception of items handled by the Picture class, running them with the various language options doesn't result in any visible changes. Pictures *do* change, since they are drawn from the locale settings already installed on your machine.

The downloadable ProDomus Translator Plus demo, on the other hand, does have quite a few translation strings pre-loaded, so I would recommend taking a look at that for some instant visual translation gratification.

Technical Support

I tried the support via email, and never failed to get a very complete response in a timely manner. As a longtime user of other ProDomus products over the years, I can also verify that ProDomus has always maintained this high level of support.

Given the confusion that is possible when first approaching the subject of internationalizing your applications, it is good to know that ProDomus is there, ready to help.


Summary





Whew. Internationalization is a big topic (global, in fact), and there really is a *lot* of

planning and effort that goes into creating a truly workable, translatable application. Any tool that substantially reduced that time and effort would be tops in my book, and Translator Plus certainly seems to fit the bill.

The Translator Plus components add up to an effective, powerful combination. The fact that you can pick and choose between the components and utilities is an added bonus, as you can assemble just the tools you need for your particular needs. It is a case, though, where each additional component does add to the overall power and ease of use.

All in all, I'd say that the full ProDomus Translator Plus is the premier internationalization toolset, bar none.

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

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

*As noted above, source is provided for all class components except the Picture class. Source is not provided for the utility programs.

ProDomus Translator Plus is Clarion 5 (or higher), ABC only. If you need Legacy support, check out their PD Translator product.

Pricing for individual Translator Plus components ranges from \$49 to \$149 (all prices in US dollars). Bundles are available ranging from a Basic Edition (Picture and String Class) for \$249 up to a Professional Edition for \$779 that includes everything discussed here except the Type class. An Enterprise Edition is also available for \$899 that includes all items PLUS the original PD Translator (for use in translating Legacy template applications). Discounts are available for PD Translator users who want to upgrade to PD Translator Plus.

For exact pricing information, a downloadable demo and more, please visit the ProDomus web site at <http://www.prodomus.com>.

Vendor Comments from Phil Will of ProDomus:

Thanks, Tom, for the excellent and thorough review. For those interested in international development standards and references, there are a number of links on our web site. In particular, anyone starting an application that might be internationali(z/s)ed in the future should at least look at the draft standards at <http://www.prodomus.com/developm.htm>. The class removal template can be found at <http://www.prodomus.com/translatorplus.htm>. There is also free "open source" pre Clarion 5.5 leap year date fix function to be found at <http://www.prodomus.com/pddt.htm>.

If you have questions or comments, or are interested in having your product reviewed, you can reach Tom Hebenstreit at reviews@clarionmag.com.

[Product Review: ProDomus Translator Plus](#)

(Dec 14, 1999)

[The Novice's Corner: Understanding Clarion Code](#)

(Dec 14, 1999)

[Open Source Update: Date Fix & DDE Classes](#)

(Dec 14, 1999)

Copyright © 1999-2000 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.

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.²⁵/month**[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)

Feature Article

December, 1999

Understanding Clarion Code Part 2

by David Harms

In the [first article in this series](#) I discussed the basic format of Clarion source code and how the project system turns this into a working program. In this article I'll look in more detail at the language and discuss a small utility program for deleting old Windows temporary files.

But first, a few details about data types and operators. If you're familiar with these you may want to skip down to the [example program](#), but then again if you read through you may just learn a few things that surprise you.

Data Types

In the previous article I touched on how Clarion allocates memory to variables, and how data declaration affects scope, or visibility. The following is a discussion of what form these declarations can take.

Clarion data declarations can be loosely grouped into three kinds: simple, complex (or structured), and special.

Simple data types

Clarion simple data types include BYTE, SHORT, USHORT, LONG, ULONG, SIGNED, UNSIGNED, SREAL, REAL, BFLOAT4, BFLOAT8, DECIMAL, PCDECIMAL, STRING, CSTRING, PSTRING, DATE and TIME. And there are a couple of new types coming in Clarion 5.5 including ASTRING and BSTRING.

[Click here for a list of Clarion data types and their restrictions.](#)

Some example variable declarations:

```
bVar    BYTE
lVar    LONG
sVar    STRING( 20 )
```

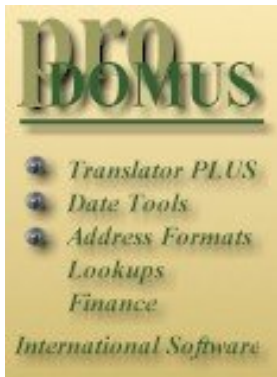
[Understanding Clarion Code Part 1](#)

[Product Review:
ProDomus Translator Plus](#)
(Dec 14, 1999)

[The Novice's Corner:
Understanding Clarion
Code](#)
(Dec 14, 1999)

[Open Source Update:
Date Fix & DDE Classes](#)
(Dec 14, 1999)





It's vital to know the limitations of the data types you're using. For instance, suppose you're using a `BYTE` variable called `X` as a loop counter, and you have code something like this:

```
LOOP X = 1 to 300
    ! do something
END
```

This code will run forever because whenever `X` reaches 255, the next increment will take it back to zero, and the value of 300 will never be achieved.

Another place incorrect data types can cause problems is on Windows API calls where an unsigned variable is expected and a signed variable is used. Windows will load up the specified number of bytes (i.e. two bytes for a `USHORT`) with value, but your Clarion program will interpret those bytes as a `SHORT` resulting in a quite different number than the one Windows stored.

You also need to be aware of when an integer variable is required, and when a floating point or decimal variable is required. If you attempt to store a floating point or decimal value in an integer variable, the decimal portion will be truncated. You don't have to explicitly tell Clarion when you're changing data types – you can just say (assuming you have a `LONG` variable called `LongVar` and a `REAL` variable called `RealVar`):

```
LongVar = RealVar
```

and Clarion will handle the data type conversion for you. It'll do numerics to strings and the other way around (if the string contains numbers, that is). But because Clarion does the type conversion you have to be extra careful that you're using data types that won't cause a loss of data (unless that's what you want).

Many of the data types in Clarion are there to support either particular types of databases or Windows API functions. `SIGNED` and `UNSIGNED` are useful with the latter but aren't actually data types. They're just assigned (using `EQUATE`) to the appropriate data type depending on whether the code is 16 or 32 bit.

Some of the more commonly used simple data types are `BYTE`, `SHORT`, `LONG`, `REAL` and `DECIMAL`. For floating point calculations, `REALs` are much faster than `DECIMALs`, but they lack some precision (this is a result of how the hardware does floating point calculations, not a shortcoming in Clarion). Where precision is essential you should use `DECIMALs`. When `DECIMALs` are used instead of `REALs` in a calculation Clarion invokes its internal binary coded decimal (BCD) library, which is accurate to within 31 decimal places on both sides of the decimal point. There are several rules governing when the BCD library is invoked. See the online help for BCD for more information.

`PDECIMALs` are mainly provided for `Btrieve` compatibility and are not otherwise much used.

Numeric variables can be declared with initial values, as in:

```
MyVar LONG(447)
```

In this example `MyVar` will be initialized to 447.

Among string data types the two most-used are `STRING` and `CSTRING`. `STRING` is the string type that has existed in Clarion since its origin, and `CSTRING` is the C language equivalent. The difference between the two is that `STRINGs` are padded with spaces, but `CSTRINGs` terminate with a null character. Say you have two `STRING` variables of 20

characters, and the first is set to 'John' and the second to 'Smith' and you want to combine them into a third STRING variable, which is 30 characters long.

```
FirstName      STRING(20)
SecondName    STRING(20)
FullName      STRING(30)
```

```
CODE
FirstName = 'John'
SecondName = 'Smith'
FullName = FirstName & ' ' & SecondName
```

The problem is that FullName will end up looking like this:

```
John Smith
```

In a STRING spaces are valid characters, and have to be stripped using the CLIP() function:

```
FullName = CLIP(FirstName) & ' ' & CLIP(SecondName)
```

If, on the other hand, you use CSTRINGs instead of STRINGs you don't have to do the CLIP at all. In the code:

```
FirstName = 'John'
SecondName = 'Smith'
```

FirstName will have a null character after the letter n and SecondName will have a null character after the letter h. This is done automatically whenever a value is assigned to a CSTRING; the null specifies where the string ends.

So why would anyone bother with STRINGs? For some of us, it's just habit. And with CSTRINGs you always have to allocate one more character than you would normally, because the last byte will always be a null value. So the declaration would look like this:

```
FirstName      CSTRING(21)
SecondName    CSTRING(21)
FullName      CSTRING(31)
```

That may not seem like a bit deal (what's an extra byte) but just remember it when you're dealing with strings of just a few characters!

Dates And Times

DATE and TIME are four byte values for dates and times, and are standard data types in SQL databases. Clarion applications traditionally have used LONG fields for dates and times, in the standard Clarion format. Clarion standard dates are stored as the number of days since Dec 28, 1800, and standard times are stored as the number of hundreds of a second since midnight, plus one. (Because of that extra hundredth of a second, time calculations can be tricky. [Click here for more](#). Clarion standard dates and times (look for that phrase in the online help) stored in LONGs are fine if you're strictly using Clarion to access your data, since Clarion knows how to format the data. But if you'll be using any other products, be warned that most people will not be able look at the raw file data and know, for instance, that 72350 translates to February 8, 1999. (In fact it doesn't translate to that date at all. Get the point?)

Arrays

If you want more than one copy of a particular variable, you may find it easiest to use arrays. When you declare a variable use the DIM attribute to specify how many elements you want in the array. For an array of 20 name fields, you could use something like the following:

```
NameField STRING(30),DIM(20)
```

To reference the individual elements of the array you need to specify the array index in square brackets. To set the fourth NameField in the array you'd use the following:

```
NameField[4] = 'Some Value'
```

Arrays are a powerful addition to variable declaration, and begin to blur the line between simple and complex data types. In Clarion arrays always begin with the index 1 rather than zero. If you do attempt to access the zero element of an array you'll be attempting to use memory that doesn't belong to the array, and that can have all kinds of bad results.

Clarion will only tell you that you have an out of range array index if you have debug on and Array Index checked in your debug settings. In 16 bit code you'll get a compiler error if you're using an out of range constant to check the array index, and a runtime error in all other cases.

In 32 bit code you will get these errors if the subscript is greater than the array size, i.e. you attempt to access element 6 of a DIM(5) variable. However in 32 bit standalone apps there is NO check, runtime or compile time, on accessing the zero element of an array! So be very careful if you're coming from a language that uses zero-based arrays..

Complex Data Types: GROUPS

Complex data types are (generally speaking) combinations of simple data types. One of the most common is the GROUP. Here a small example:

```
MyGroup    GROUP
Var1       STRING(20)
Var2       STRING(20)
          END
```

GROUPS are a handy way of associating data types so you can move, copy, or otherwise deal with them as a unit. The most-used variation on this theme is the file RECORD structure which is really just a grouping of the fields in a file.

QUEUES

Another oft-used data type is the QUEUE. A QUEUE is something like an array, but you don't specify ahead of time how many elements, or records, a QUEUE can have.

```
MyQueue    QUEUE
Var1       STRING(20)
Var2       STRING(20)
          END
```

One of the great conveniences of Clarion over the years is its ability to clean up after itself. In lower level languages you the programmer have to explicitly allocate memory for variables. Clarion always takes care of allocating memory for, and cleaning up, simple data types (except for those you create yourself using NEW, but that's another subject). QUEUES are slightly different. To add a record to a QUEUE you set the values

of the fields and then do an `ADD(queueName)`. When you're done with the `QUEUE` you should issue a `FREE(queueName)` to release the memory taken by `ADD`. Procedure level `QUEUES` will have their memory cleaned up when the procedure exits, but module and global `QUEUES` will use up memory until the program exits. So clean up after yourself.

`QUEUES` are quite powerful. You can sort on any field or combination of fields and retrieve matching records, and you can have `QUEUES` of just about every type of data. I'll be coming back to `QUEUES` later in this article.

CLASSES

The `CLASS` structure deserves a whole series of articles on its own, and is well beyond the scope of this article. If you'd like more information on how Clarion `CLASS` structures enable object-oriented development, see the [ABCs of OOP](#) series of articles.

Special Data Types

Clarion has a number of special data types, including reference variables (a sort of safe pointer), and the `ANY` data type, which can be used to store any numeric or string value or a reference to any simple data type. Feel free to read up on these subjects in the Language Reference – they won't be part of this series for a while yet.

Finally, An Example!

Here's an example, provided by Carl Barnes, that demonstrates some of the basics of Clarion data declarations and also introduces some of Clarion's program flow control statements. This is a little utility which cleans out Windows temporary files that are more than four days old.

Figure 1. A utility program to delete old Windows temp files.

```
PROGRAM

    MAP
    END

DirQ      QUEUE(ff_:Queue),PRE(AnythingAtAll)
          END

Count     LONG
Idx       LONG
TempDir   STRING(255)

CODE
TempDir = 'C:\windows\Temp'
!TempDir = 'C:\Temp'
Count = 0
DIRECTORY(DirQ,CLIP(TempDir)&' \*.*',ff_:Normal)
LOOP Idx = 1 TO RECORDS(DirQ)
    GET(DirQ,Idx)
    IF DirQ.Date < TODAY() - 4
        REMOVE(CLIP(TempDir)&' \ '&DirQ.Name)
```

```

        Count += 1
    END
END
MESSAGE( 'Done! ' & Count & ' file(s) removed' )

```

The beginning of this program should look fairly familiar if you've read the [first article in this series](#). But the first data declaration may be a bit of a puzzler.

```

DirQ    QUEUE(ff_:Queue),PRE(DirQ)
    END

```

Clearly this is a `QUEUE` declaration, but it's of a slightly different form. Clarion allows you to build complex data types based on other data types. One way to do this is with the `LIKE` statement; the other, which applies to `QUEUES` and `GROUPS`, is to use an existing `QUEUE` or `GROUP` as a parameter to the declaration. `DirQ` is based on something called `ff_:Queue`, and you can find out more about `ff_:Queue` in the Clarion help under `Directory`. Or you can just take it on faith that this is a `QUEUE` with some predefined fields.

So where is `ff_:Queue` declared? You may remember from the previous article that the `MAP` statement has the hidden benefit of automatically `INCLUDEing` `EQUATES.CLW`, which is in your `LIBSRC\` directory, and that's where you can find `ff_:Queue`.

You can also, for the moment, ignore that strange `PRE` attribute. I'll come back to it shortly.

Following `DirQ` are some fairly normal looking variable declarations:

```

Count    LONG
Idx      LONG
TempDir  STRING( 255 )

```

Next is a `CODE` statement indicating the start of the executable part of the program. Several variables are initialized (you'll want to set `TempDir` to whichever directory your installation of Windows uses for its temporary files).

```

CODE
TempDir = 'C:\windows\Temp'
!TempDir = 'C:\Temp'
Count = 0

```

Now things get interesting. `DIRECTORY` is a library function that retrieves a directory file listing.

```

    DIRECTORY(DirQ,CLIP(TempDir)&' \*.*',ff_:Normal)

```

The first parameter is the `QUEUE` that will hold the file listing. This `QUEUE` has to be in a specific format, which is why it's based on the definition in `EQUATES.CLW`. The second parameter is the directory to retrieve, and the third parameter is a flag that tells the function what kind of files to retrieve. These flags are defined in `EQUATES.CLW` as well:

```

ff_:NORMAL          EQUATE( 0 )

```



```

ff_ : READONLY      EQUATE ( 1 )
ff_ : HIDDEN        EQUATE ( 2 )
ff_ : SYSTEM        EQUATE ( 4 )
ff_ : DIRECTORY     EQUATE ( 10H )
ff_ : ARCHIVE       EQUATE ( 20H )
ff_ : LFN           EQUATE ( 80H )

```

EQUATE is a compiler directive which just means that a given label (such as `ff_ : ARCHIVE`) should always be interpreted as the specified value. These flags are all defined so they can be used as bitwise flags, so they can be added if you want to combine features. For instance, to get a list of all of the normal and read-only files you would pass `ff_ : NORMAL + ff_ : READONLY` to the `DIRECTORY` procedure.

Although it may not look like it, the three parameters being passed to the `DIRECTORY` procedure aren't all handled the same way. Simple data types can be passed one of two ways; by value or by address. When a parameter is passed by value, the receiving procedure makes its own copy of that data. Any changes the receiving procedure makes to that data isn't reflected back in the calling procedure.

When data is passed by address, any changes the receiving procedure makes to the data *are* reflected in the calling procedure. For simple data types, a `*` is used in the prototype to indicate when something is passed by address.

The prototype for `DIRECTORY`, and all the other library functions, can be found in `LIBSRC\BUILTINS.CLW`:

```

DIRECTORY(ff_ : queue result, STRING path, BYTE attrib), ←
          NAME( 'Clas$DIRECTORY' )

```

You don't see any `*` in that prototype. Look a little further down the file however and you'll note that some prototypes do pass data by address:

```

WHAT( *GROUP, SIGNED), *?, NAME( 'Clas$WHAT' )

```

So is `DirQ` passed by value? That's a trick question. Certain data types are defined as *entity-parameters* and are *always* passed by address. `QUEUES` are one such data type. For more information see the help under `Prototype Parameter Lists`. This is one of the most vital pieces of documentation that comes with Clarion.

After `DIRECTORY` completes, `DirQ` will contain a directory listing which matches the attribute flags you specified. The `QUEUE` structure looks like this:

```

ff_ : queue      QUEUE, PRE( ff_ ), TYPE
name             string( 13 )
date            long
time           long
size           long
attrib         byte
                END

```

(A digression: You'll notice that in the text of this article I'm using all caps for Clarion keywords, in keeping with stated Clarion practice. Yet the `ff_ : queue` declaration is all lower case. That's what happens when you try to get programmers to be consistent. Personally I prefer lower case and mixed case to all caps, but for the sake of readability I've gone with the Clarion convention for this series of articles.)

This form of the `QUEUE` deals only with short file names – there is another for long file

names. The fields include the file name, date and time stamps, the size, and the file attributes, corresponding to the equates listed above. In particular the date and time have been converted to Clarion standard date and time formats which makes an aging test very easy.

The code that deletes the files makes use of two program flow structures: LOOP and IF. The LOOP structure can have several forms, and in this case will loop continually beginning with the initial condition `Idx = 1`, and continuing until the condition `Idx = RECORDS(DirQ)` is met.

```

LOOP Idx = 1 TO RECORDS(DirQ)
  GET(DirQ,Idx)
  IF DirQ.Date < TODAY() - 4
    REMOVE(CLIP(TempDir)&'\'&DirQ.Name)
    Count += 1
  END
END

```

`Idx` is incremented on each loop iteration, and is used as an index to retrieve directory QUEUE records. When used this way QUEUES are very similar to arrays, as each QUEUE record has an internal record number starting at 1.

Each file's date is tested to see if it is older than four days:

```
IF DirQ.Date < TODAY() - 4
```

Because the `Date` field is a member of `DirQ` it can (and I think should) be referenced as `DirQ.Date`. This "dot notation" is common to object-oriented languages, and Clarion is becoming more OO with each release.

The more traditional approach is to use a prefix, as indicated by the `PRE(DirQ)` attribute. When you specify a prefix the compiler automatically adds a colon to the end of the prefix text. You can then refer to the `Date` field in the QUEUE as `DirQ:Date`. The only difference in this example is the colon instead of the dot.

In fact, if you were working with a straight QUEUE declaration instead of one that was *derived* from `ff_ :Queue`, you wouldn't need a prefix at all. But you do need one because of how the compiler handles the naming of GROUP and QUEUE fields internally. It's a little complication that only affects derived QUEUES and GROUPS. Remove `PRE(DirQ)` and compile and you'll see what I mean.

Prefixes are also typically used on file layouts. And although they're a bit of a throwback, they do have two big advantages: they're short, and they ignore nesting. Say you have a GROUP called `People` with a prefix `PEO`, and it looks like this:

```

People      GROUP , PRE ( PEO )
FirstName   STRING ( 30 )
LastName    STRING ( 30 )
Phone       GROUP
Area        SHORT
Number      LONG
            END
            END

```

You can use `PEO:Area` to refer to the area code. But if you want to stick with the dot syntax, you'll need to say `People.Phone.Area` which is a bit more of a hassle.

Without the prefix you have to fully qualify the field names with any nested structures.

Back to the code. If the file in question is more than four days old, it's deleted using the REMOVE library function. Because this is a non-critical function there's no error checking; it isn't a big deal if a temporary file can't be deleted.

The last task is to display a message letting the reader know what's happened.

```
MESSAGE('Done! ' & Count & ' file(s) removed')
```

And that's it. You now have a handy utility program for removing unneeded temporary files. Thanks, Carl!

Summary

It's not difficult to write small utilities like this one, but then again there's a lot missing from this utility too. Almost all applications display windows and respond to events, behavior that is absent from this example. In the next article I'll introduce the Window structure and show how to add a user interface to the directory cleanup program.

[Download the source code](#)

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

[Product Review: ProDomus Translator Plus](#)

(Dec 14, 1999)

[The Novice's Corner: Understanding Clarion Code](#)

(Dec 14, 1999)

[Open Source Update: Date Fix & DDE Classes](#)

(Dec 14, 1999)

Copyright © 1999-2000 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.

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.²⁵/month**[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)**Feature Article**

December, 1999

ABC Embeds Are Easy!

By Tom Giles

I started using Clarion Professional Developer (CPD) 2.0 with Designer when it first came out in the early 80's. I found it had a fairly steep learning curve for a DOS product but once I understood it there was nothing I couldn't do, even though much hand coding was usually necessary.

When I finally had the time and need to move into Windows I started with Clarion4. I was overwhelmed as most users seemed to be. I looked at the embeds of one of my forms and found about two thousand possible entry points ([click here](#) for a partial listing screen shot). I had no idea where to start or how to begin to understand them. I never heard a clear explanation of embeds. It was as if no one really knew or if they did, they wanted to keep it a secret.

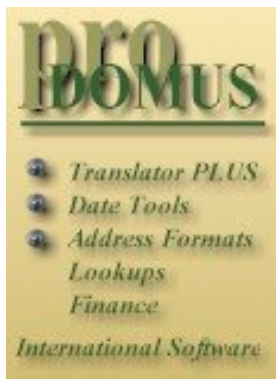
Over time I have gotten to the point of feeling I have a good understanding of the ABC embeds. And once you know what embeds are for and how they work, they're really quite easy to use. This article will discuss embeds based on a form procedure, since that is where the majority of them will probably be needed.

First of all embeds allow you to enter your personal code before or after each block of code (usually class methods in ABC) written by members of the TopSpeed Development Centre in London, England. The people in London do the hard work and optimize and debug the complicated code. You do your part to make it fit your program needs. It's really a very good arrangement.

In Legacy Clarion, the embed names were directly tied to the template so they generally followed the program flow in a logical start to finish order. They had specific and meaningful names even a beginning programmer could understand. ABC (Application Builder Classes) introduced OOP (Object Oriented Programming) which is based on a series of "black boxes" of small code snippets of debugged, highly efficient code. Since you can't easily get to that code, and probably wouldn't want to if you could, embeds allow you the programmer to put your code before or after the class's (or object's, if you prefer) code.

This arrangement offers a great deal of flexibility but at the same time gives an

[ABC Embeds Are Easy](#)
(Dec 21, 1999)[December 1999 News](#)
(Dec 21, 1999)[Edit-In-Place CheckBoxes
Done Right](#)
(Dec 21, 1999)[A Class Wrapper For Files](#)
(Dec 21, 1999)[Clarion Magazine Holiday
Schedule](#)
(Dec 21, 1999)**TopSpeed****Clarion 5**
by TopSpeed

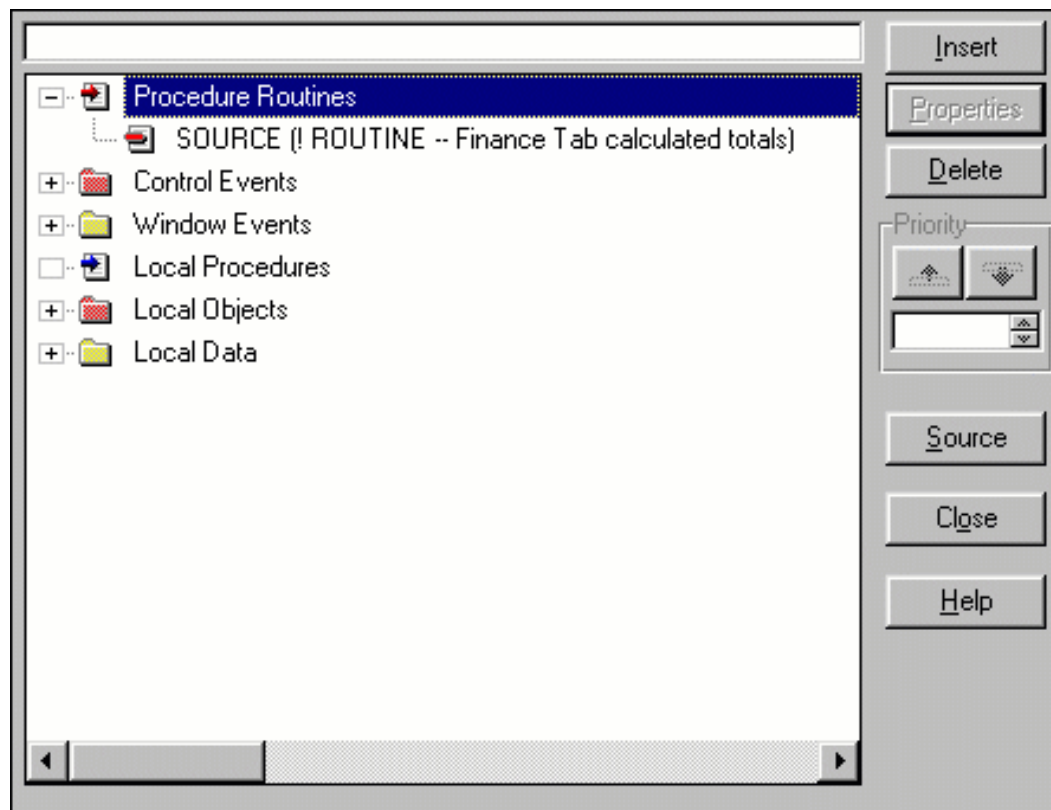


overwhelming number of possibilities. The embed names are purposely fuzzy (vague or cryptic) since the code objects could be used many different places for many different purposes. The upshot is the naming convention is actually logical but at the same time confusing, especially to the new programmer. Perhaps as ABC matures two sets of names could be evolved. One would be the strictly ABC names, the second set template-oriented similar to the Legacy embed names. This might also be an avenue (read revenue) for third party developers. The Clarion 5 Learning Your ABC's handbook has a partial listing that is helpful.

Not to be too nostalgic, but it was fairly simple in DOS. There were three major embed points or lines. Procedures had a Setup line, a Next line and edit lines for each data entry field. The Setup line allowed initialization entries for the procedure and the Next line allowed post procedure processing. The edit lines were for field validation and error checking as well as preprocessing for the next field. Remember that unlike Windows the programmer was in charge and specified the movement throughout the form. In Windows the users are in charge and can and will do as they desire when it comes to navigating the form.

OOP is also very simple at its heart but allows/creates the mega-multitude of embed points since you can add your code before or after each of the ABC code snippets from London. This is both an advantage and disadvantage. It means your hand coding will be minimized which should result in less work and bugs. The disadvantage is not knowing where to start.

Figure 2. The embed tree (collapsed except for one routine embed).



Procedure Routines (Routines)

I'll start with the easy one at the top, the Procedure Routines. This is just what it says, a place to put Routines for this procedure. Basically a Routine is a section of code that will be used a number of times within a procedure. Instead of embedding (placing) the same

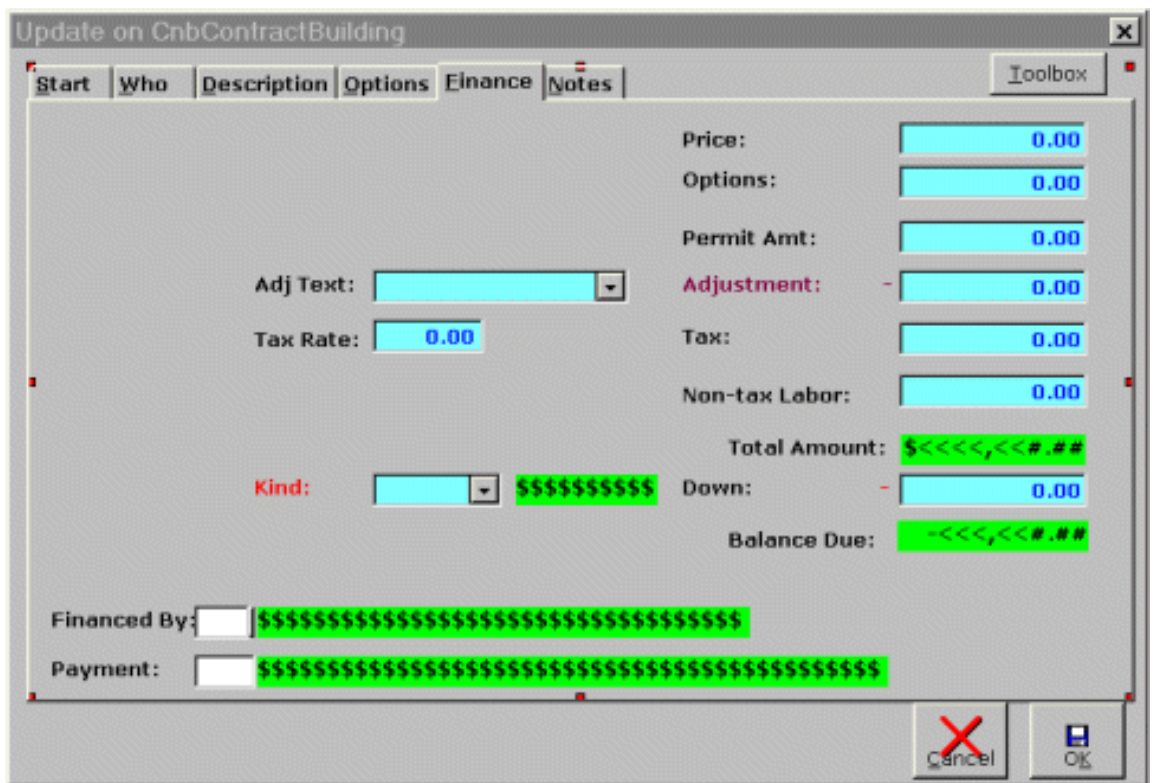
piece of code multiple places throughout the procedure, you put it in one place and just call it repeatedly as needed. This allows for easy maintenance as the code is in one location and is written only once. Notice the similarity to OOP. And you thought the concept was difficult.

In CPD the Model files did not support Routines so you had to do extra hand coding. Not bad once you figured out how but not as easy as just dropping the code in. The basic syntax for a routine is:

```
RoutineName ROUTINE
!Your code goes here
```

An END statement is not needed for the ROUTINE statement. To call the routine, just enter DO RoutineName at the appropriate embed points and the routine will execute. For an example look at the Finance tab in Figure 3.

Figure 3. A form in need of embedded code.



Here I want to calculate the Tax, Total Amount and Balance Due any time a change is made to any of the listed numbers as well as the PayoffDate and some others, (which are shown on other tabs). In CPD I would have used a computed field which would do the three calculations every pass through the Accept Loop (except of course it wasn't called an Accept Loop then). Similarly, the Clarion5 Formula Editor will do calculations at specific points as defined in the Class entry point. You could use one of the embed points such as TakeFieldEvent (it goes through this embed after every field acceptance) but the code would be somewhat buried amongst the others and would be hard to find if you needed to edit it. A Routine is good here. [Click here](#) for a (large) screen shot of my routine code.

For each of the fields at the right of the screen in Figure 3, and the Tax Rate, I enter DO Finance in the ControlEvents|?datafieldname|Perform field level validation embed point. Since the Price and Options came from previous tabs on that window I put a DO

Finance at those points also. This way any time the user looks at the Finance tab it is correct.

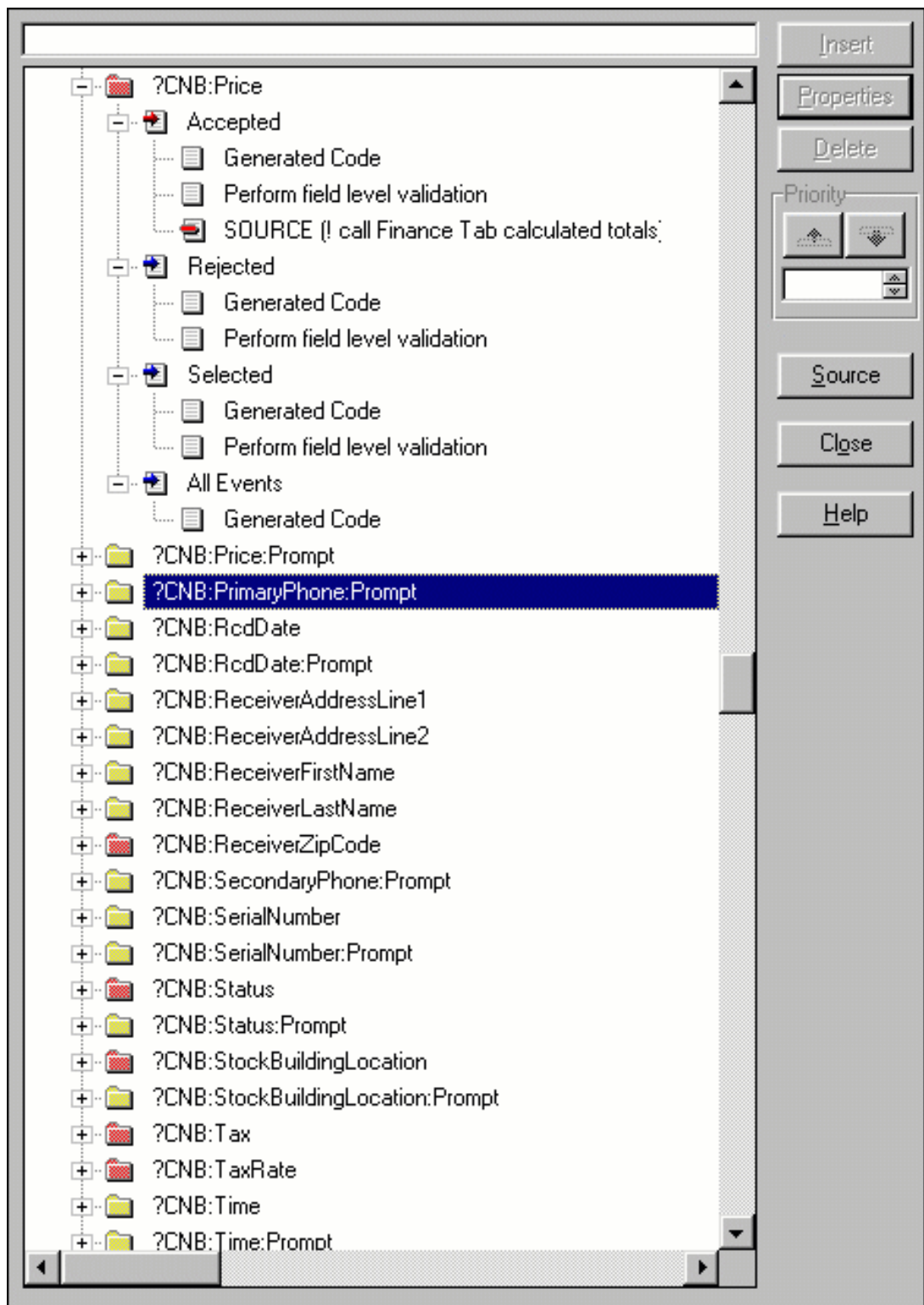
Several other points about the above code you might want to consider: My top line of any embed is a one line explanation of the purpose of the embed for quick reference (i.e. six months from now). I also use a lot of white space in my formulas (formuli to be correct) since I find them easier to read. For the END statement I copy the corresponding starting line as a comment so there will be no doubt what it is for. This is helpful when there are several nested statements each with a corresponding END. I still use THEN and a period (.) in lieu of END as a carryover habit of my DOS days on occasion. You need to develop your personal programming style then stick with it.

Remember programming is basically logical thinking and attention to detail. A good style helps prevent errors and makes a later review of your code much easier. I also believe in a lot of comment statements, but not of the very obvious code. Stress the purpose or the why; don't just repeat the code a second time.

Control Events (Data Field Editing)

Referring back to Figure 2 you'll see that the next section in the embed list is Control Events. This is equivalent to the CPD Edit line. Here you put your code to validate the current field or preprocess the next field. Figure 5 shows the expanded list of embeds for ?CNB:Price.

Figure 5. An expanded Control Events embed list.



In Figure 5 you see the embeds for the Price field. I have my one liner DO Finance in the Accepted|Perform field level Validation|SOURCE embed. The Perform field evaluation is the one you will use the most often. The others may be needed under special circumstances but I very seldom use them. Note that you will not have to write the setup line of code and the END statement since the various possibilities (accepted, selected, etc.) are already set up for you. Just put the meat of the code in the appropriate place. It's another simplification for you.

I tend to use a lot of these embed points for error checking of data entries. Do all that you can to insure only good and valid data gets in the system as it will make your job much easier. I also tend to put a lot of error messages here (using the MESSAGE ()

function) to clearly explain what is wrong with the input so there will not be any confusion. You can't force the user to read the messages but try and let them know anyway.

There are two embeds you should not use here: they are ?OK and ?Cancel. If you embed code in these and the users clicks the **X** in the upper right corner of the screen then the Yes or No message about saving the Record your OK/Cancel code will be missed and users will complain and potentially bad things could happen or not happen depending on your code. See Local Objects|ThisWindow below for a solution.

Window Events (Advanced)

The next section is Windows Events. This allows embeds about the screen and the window itself. This is for special situations and probably won't be used that often, at least not by the beginning programmer. It's nice to have when you do need it, however.

Local Procedures (Advanced)

Local Procedures is another special use case more for the advanced user with special needs.

Local Objects (ThisWindow)

Local Objects is very intimidating for the novice. Your main concern will be with ThisWindow. If you use the toolbar you might be concerned with that also. Consider anything else here special cases for the advanced user and disregard for now. Even so ThisWindow is still very intimidating by itself. But there are only a few embeds you will need to be primarily concerned. `Init` is similar to the Setup line in CPD. In CPD 2.1 the files were opened as the program started. In Windows Clarion they are opened/closed as needed. Take this into account when embedding code in `Init` or `Kill`.

`Kill` is analogous to the CPD Next line. It will close all the files listed in the Files portion of the Procedures screen and allow for any post-processing code. The `TakeCompleted` embed should be used for your ?OK code as it will be triggered for the OK button as well as the **X**\Yes. The `TakeCloseEvent` (after the parent call, which is the Save This Record box) is the best place for the Cancel button and **X**\No code. See my upcoming article titled "An ABC Gotcha" for more information on this subject. `TakeEvent` is similar to the CPD Accept loop and will be triggered throughout the program. Some of the other `Take?????` Embeds may be needed on occasion.

Local Data (Advanced)

Local Data is another section you probably won't need. Use the Data section at the Properties screen instead.

Self Help

In trying to learn embeds I have tried several methods and techniques. I started by using the brute force approach of putting `STOP` statements in any embed that looked remotely usable, and even some others for good measure. I used `STOP(1)`, `STOP(2)`, `STOP(3)`, etc. then ran the program making notes of what came up and when. This method works okay most of the time. You are good to the first `STOP` statement. Since

you must respond, a message is sent to Windows which could be misinterpreted and cause unexpected results. Normally this is not a big problem but it has happened to me.

The technically better approach is `MESSAGE (1)` , `MESSAGE (2)` , etc. This should not confuse Windows as much as the `STOP` statement does. One advantage of the `STOP` statement is that it will allow Abort for an easy out. The `MESSAGE` statement can create an endless loop. I start with the `STOP` then maybe go to `MESSAGE`.

Another approach is to click on the red Legacy shield (see the tool bar in Figure 1 between the two activated buttons) to bring up the English-like Legacy embed names and chose from there. Write something such as `STOP (embed name)` then click off the Legacy shield to go back to ABC. Right click on the procedure and click on Source and do a search on `STOP (` to find your code. Move it up or down to the nearest ABC embed and erase the temporary Legacy embed.

Finally, the Debugger is not the easiest or quickest to use but can be instructive. ([Click here](#) for more on the debugger.) Click to halt on each code statement in the source. The "weird" code that goes by between each halt is the London code and is found in the `\Clarion5\Libsrc` directory. It's interesting, but also confusing, especially at first.

One More Technique

Another technique is to break a long embed into smaller segments (see Figure 6). As an example I have a Daily Report that is made of 10 separate reports. Five of these reports require reading data from 11 separate files. Needless to say the setup would be a very long embed. It would also be confusing while writing it, not to mention looking at it six months from now. I made a separate embed point for each complicated report, for a total of seven embeds plus the summary as shown.

Now the embeds dealt with smaller cohesive data with the same purpose in mind. The top line of each embed was the same with – sub-report name added at the end. To keep them in proper order I used the original priority for a dummy embed of just comments explaining what I would do. I added 1 to the priority for each of the others. This means they will be executed in the proper order and the sequential numbers (by 1) from 7600 to 7607. The numbers are another indication of a group that goes together. If other embeds were needed here I'd start another group of priority numbers. I find this technique makes code much more readable and manageable.

[Click here](#) for a screen shot showing prioritized source embeds.

Summary

That's basically it for ABC embeds. Use Routines if needed. Use Local Events for error trapping and field validation. Use a few selected Local Objects|ThisWindow embeds for initialization and getout (next) procedures plus a couple of others as needed. These few embeds are a good start and with time you may find the need for the multitude of other embed points. Start simple and investigate. Experiment and grow as you gain experience and confidence, and your needs expand. As I said at the start, ABC embeds are easy!

[Tom Giles](#) is a self-taught, long time CPD 2.1 programmer who started with interpretive BASIC for his business programs. He is moving his CPD programs to ABC for the long haul. When not programming he is out skydiving or flying his homebuilt airplane. Isn't life grand?

[ABC Embeds Are Easy](#)

(Dec 21, 1999)

[December 1999 News](#)

(Dec 21, 1999)

[Edit-In-Place CheckBoxes Done Right](#)

(Dec 21, 1999)

[A Class Wrapper For Files](#)

(Dec 21, 1999)

[Clarion Magazine Holiday Schedule](#)

(Dec 21, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resources

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.²⁵/month**[Main Page](#)[Log In](#)[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)[Article Index](#)[Author Index](#)[Links To](#)[Other Sites](#)[Downloads](#)[Open Source](#)[Project](#)[Issues in](#)[PDF Format](#)[Free Software](#)[Advertising](#)[Contact Us](#)

News

December, 1999

Clarion News

December 21, 1999

[Updated Clarion 5.5 Resources](#)

The TopSpeed Resource Center contains a variety of Clarion 5.5-related information including an example web app, ISAPI configuration, feature list, and web styles and TSScript documentation.

[TopSpeed ISP E-Commerce Support Information](#)

TopSpeed has a page for ISPs interested in supporting Clarion web applications and participating in its WiredOnMainStreet e-commerce initiative. Interested parties can visit the page or call TopSpeed at (800) 354-5444 ext. 200 or (954) 785-4556.

[LSZip Backup/Restore Demo](#)

A new LSZip Backup/Restore source code demo is now available from Linder Software. Compatible with Clarion versions 4, 5, and 5.5. LSZip manages industry standard ZIP (100% compatible with PKZIP) and LSP archives and supports PKZIP compatible disk-spanning, encryption and in-memory compression.

[MessageEx Update Available](#)

MessageEx 1.1 is now available. Features include sound support, charset support, flat buttons, improved default button handling, and a system menu. MessageEx is 39 EURO until December 24, 1999, after which it will 49 EURO. This update is free for all registered users.

[CCS/SQL 5A Update for ABC](#)

Version 5A of the CCS SQL template set is now available, and is a free upgrade for all current users. The install requires your version 5 password and an unlock code returned by email after your order and license are validated. Click "Secure online ordering" on the web page, and on the following form check the \$0.00 Update Only entry field.

[SysAni Update Available](#)

SysAni 1.1, an animation control, is now available. This version fixes a bug in the templates and contains files for Clarion 5.5 beta1. The update is free for all registered users.

December 14, 1999

[TopSpeed Releases ODBC Driver Update](#)

[ABC Embeds Are Easy](#)

(Dec 21, 1999)

[December 1999 News](#)

(Dec 21, 1999)

[Edit-In-Place CheckBoxes](#)[Done Right](#)

(Dec 21, 1999)

[A Class Wrapper For Files](#)

(Dec 21, 1999)

[Clarion Magazine Holiday](#)[Schedule](#)

(Dec 21, 1999)



**How to
Put More
WOW!
into your
apps**

[Click Here](#)

TopSpeed has just released an updated ODBC driver for accessing TopSpeed (.TPS) data files. This release offers major enhancements to the supported SQL syntax and is ODBC 3.0 compliant. Call TopSpeed Sales at 1-800-354-5444.

[New Third Party Accessories At TopSpeed](#)

Pea Brain Software's Documentation Expert and Localizer products are now available from TopSpeed. The Documentation Expert creates product documentation from applications and dictionaries, and the Localizer lets you add unlimited languages to your applications.

[Updated Function Declarations For SocketTools](#)

For users of SocketTools 3.x Library Edition, updated function declarations and constants for Clarion 4/5 are now available from Catalyst by ftp. Updates to the Library Edition are also available. <http://www.catalyst.com/support/stfixes.html>

[Linder SetupBuilder 3.0](#)

[Release Candidate 1 Available](#)

Linder SetupBuilder 3.0 Release Candidate 1 is available now. New features include CAB file creation, calling custom DLLs to validate user information, Clarion and Borland DLL sample code, and an updated User's Guide. Linder SetupBuilder 3.0 lists for \$119.00 USD. The update to the new SetupBuilder 3.0 lists for \$71.40 USD.

[Ragazzi Product Update Notice](#)

New from Software by Ragazzi: Toolbox template to manage toolbox events; updated Edit\View Manager, and a new Built-In Functions template which adds common functions such as decimal/hex/binary conversion.

[Subject: Freeware Barcode Template Set](#)

The WASP Barcode Support Templates are now available free of charge from Stephen Mull & Associates. This is a 32-bit template set for Clarion 5.x to support the WASP Barcode development tools from Informatics, Inc. The WASP Active X - DLL Barcode Library is required, and must be purchased separately if you do not already own it. The templates work with Legacy and ABC. anyone making improvements or additions to the product is asked to provide Stephen Mull & Associates with a copy of the new work so it can be made available to the Clarion developer community. Special thanks and credit to the very talented (and anonymous) developer who assisted with this project, you know who you are!

[BackFlash 4.0 – New Major Release](#)

Sterling Data has released version 4 of BackFlash, a backup/restore module. New features include unattended backup, extensive logging, user file selection, and user-determined restore locations. A new demo is also available.

December 7, 1999

[New Version Of TX Control Class Wrapper](#)

This updated TX control class wrapper adds embedding TX text into the Clarion reports and handling TX memo overflow. Some Clarion knowledge is required.

[ShapeMaker Released at Logic Central](#)

Logic Central's ShapeMaker changes the usual rectangular window shape to rounded corners and ellipses. ShapeMaker retains pertinent system menu functionality (such as Minimize and Exit), and comes with stock shapes and shells. Use the Polygon Designer to design custom window shapes. Some possibilities: make all windows unique or just the splash window; make your application exciting with the "game console" look; easily cast your app in the "cyber" look; develop window shapes from existing images for a thematic presentations. \$49, free demos available for download.

[Mitten Offers Verified File Transfers](#)

Designed for organizations with 10 or more remote sites or users, dataINtact delivers files to and from remote sites over the Internet to your central data repository. Data is compressed and encrypted for maximum speed and security. Senders receive a certificate of receipt that guarantees data was delivered INtact. Use requires a server-based license without any additional transaction charges. Free demo available. For details click on the Information button on the web page.

[New French Clarion Site](#)

Olivier Cretey has created a French language web site devoted to Clarion tips and programming how-to (philosophy of programming, design, etc...).

[Linder SetupBuilder 3.0 Beta 6 Patch 3 Available](#)

A new SetupBuilder patch is now available. Partial list of fixes includes replacing files currently in memory (32 bit), creating system folders, folder variables, custom DLL source code example, backup of replaced files (and rollbacks), and NT4 SP6 detection. The project file format has now also changed. Linder SetupBuilder 3.0 lists for \$119.00 USD. The update from LSP SFX-Builder 2.0 costs \$71.40 USD.

[Free POP ActiveX Control](#)

Ande Knudsen has pointed out this useful ActiveX control for access POP3 mail.

[Handy Free SQL Utility](#)

If you work a lot with SQL you may want to try WinSQL, a utility like Microsoft's ISQL/W, but which works with any ODBC data source.

[Special Christmas Offer At solid.software](#)

solid.software is offering a 20% discount on MessageEx, a message() enhancement tool. Until Dec 24, 1999 the price is 39 EURO (approx. 39 USD).

[Jeff's On Holidays \(and about time, too\)](#)

J&S Software will be closed from Dec 5 to Dec 13. Orders for In Back will be handled as usual through BMT Micro. Mailmerge orders will be filled upon our return.

[List & Label Templates Available For Purchase](#)

To all those interested in the List & Label templates, you can now purchase with VISA and Mastercard via DeveloperPlus. There are now two versions of the package available: Templates only. You must have List & Label version 6.0 to purchase these. £99 (US\$160.00); New Bundle offer. List & Label version 6.0 plus templates are £355 plus £10 p&p (saving £80 if bought separately)! (US\$568.00 plus US\$15 p&p) This price includes the ABC and Legacy versions, 32bit and 16bit.

Copyright © 1999-2000 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.

published by

CoveComm Inc.

clarion magazine

Good help isn't that hard to find.**\$6.²⁵/month**[Main Page](#)[Log In](#)[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)[Article Index](#)[Author Index](#)[Links To](#)[Other Sites](#)[Downloads](#)[Open Source](#)[Project](#)[Issues in](#)[PDF Format](#)[Free Software](#)[Advertising](#)[Contact Us](#)

Feature Article

December, 1999

Checkbox Controls With Edit-In-Place

by Pete Halsted

That's right, I'm starting this article by mentioning everyone's favorite Clarion feature. TopSpeed has come a long way with their support for edit-in-place, or EIP, and with the latest ABC templates you can do some really nice things. There are classes for drop lists, spin boxes, and even checkboxes, which are the subject of this article.

So if Clarion already provides a class to do EIP for checkboxes, what's not to like? Two things: the checkbox value isn't displayed as a checkbox, and the existing class has some cosmetic problems when editing the value.

In this article I'll show you how to display your checkbox using icons, and how to give the checkbox field a consistent appearance during EIP. If you're not familiar with EIP, I suggest you head over to TopSpeed's web site and download the "Learning your ABC's" manual and read through the appropriate section.

The first thing you need is a couple of icon files, one for when the box has a check and one for when it does not. I used an icon editing program to capture a clarion screen to create these two files (on.ico and off.ico). These are included in the [downloadable zip file](#).

Next, you need to configure the browse to display the appropriate icon. I just so happen to have a browse on a simple file that has three fields: a code, code description, and a flag to allow entries. The AllowEntries field is a byte field configured to use a checkbox, which means the data will either have a one or a zero. From the list box formatter, there are two items you need to set for the AllowEntries field:

First on the General tab set the picture to "@p p" (see Figure 1). This is a picture that will always display a blank space, thereby keeping the browse box from displaying an ugly one or zero.

Figure 1. The General tab for the AllowEntries listbox field.

[ABC Embeds Are Easy](#)

(Dec 21, 1999)

[December 1999 News](#)

(Dec 21, 1999)

[Edit-In-Place CheckBoxes](#)[Done Right](#)

(Dec 21, 1999)

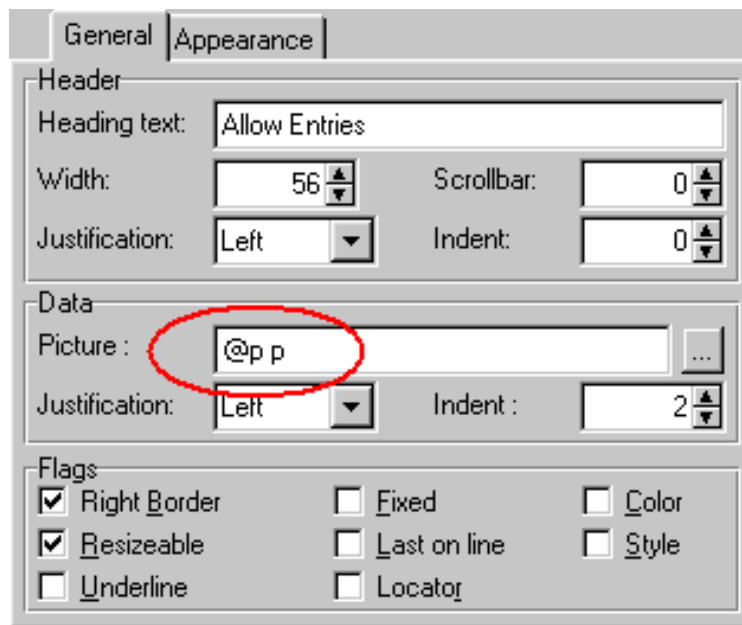
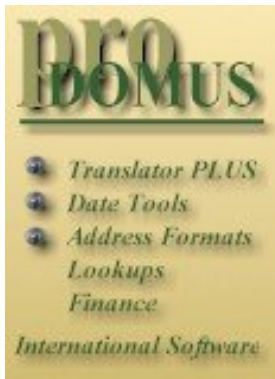
[A Class Wrapper For Files](#)

(Dec 21, 1999)

[Clarion Magazine Holiday](#)[Schedule](#)

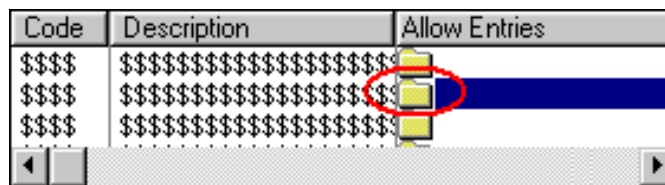
(Dec 21, 1999)

TopSpeed**Clarion 5**
by TopSpeedFREE Microsoft
Internet
Explorer 



Second, on the appearance tab, chose Normal for Icon. The browse formatter will show a folder icon to indicate that this column uses an icon. See Figure 2.

Figure 2. The listbox formatter showing a folder, which indicates the use of an icon.



By setting these two options you will enable the Icons tab on the list box control template (Figure 3).

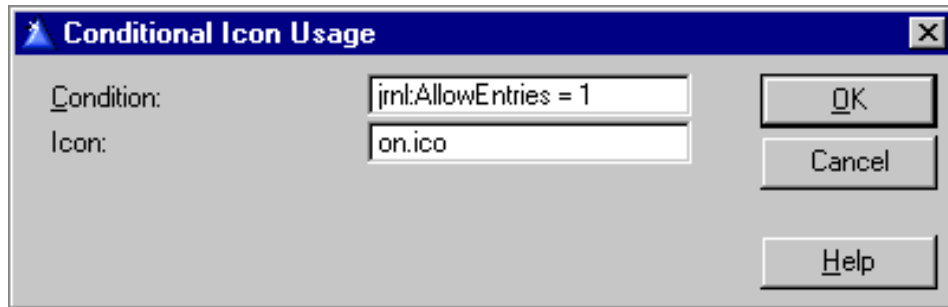
Figure 3. The Icons tab for the browse Actions settings.



There will be one entry added for the AllowEntries field. Click on Properties for this

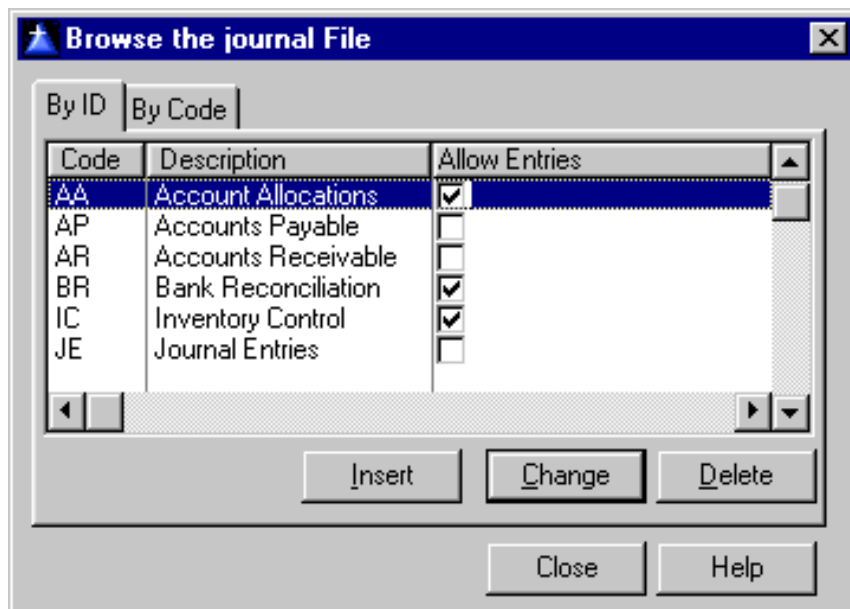
field to setup what icons to display. The first prompt is for the default icon; I always use `off.ico` as my default. If none of the other conditions I add match, then this is the Icon that will be displayed. Next, add a condition to display `on.ico` if the `AllowEntries` field is 1 (meaning the item is checked). See Figure 4.

Figure 4. Updating the conditional icon usage settings.



If you compile your application at this point you will have a list box that displays either a check box on or a check box off icon for the `AllowEntries` field (Figure 5). But wait, I promised EIP that would display correct with this, didn't I? So now on to EIP.

Figure 5. Displaying the checkbox.



The first step is to enable EIP. Right click on one of the update buttons and choose actions from the popup menu then turn check Use Edit in Place, this option is also displayed on the procedure properties screen.. Next press the Configure Edit In Place button, and then press the Column specific button from the bottom of this screen. This screen is used to override the default class setting for a specific field. Press Insert and choose the `AllowEntries` field. Removing the check "Use Default ABC" enables the class selection. From this drop down menu chose `EditCheckClass`. This is a class that TopSpeed has provided with the templates which overrides the default `EditClass` in order to create a check box control instead of an entry control.

If you compile the program at this stage you will have a checkbox for EIP, but it has a few cosmetic problems that still need to be ironed out. For instance it shows the field's picture as the prompt for the text box, and uses a gray background for the text. It also positions the check box to the right of the check box being displayed on the browse. See

Figure 6.

Figure 6. The uncorrected checkbox display during editing.

Code	Description	Allow Entries
AA	Account Allocations	<input checked="" type="checkbox"/>
AP	Accounts Payable	<input checked="" type="checkbox"/>
AR	Accounts Receivable	<input type="checkbox"/> @p p
BR	Bank Reconciliation	<input checked="" type="checkbox"/>
IC	Inventory Control	<input checked="" type="checkbox"/>
JE	Journal Entries	<input type="checkbox"/>

Clarion generates several embeds for each field enabled for EIP. There are two that are of interest; one is under `BrowseEIPManager` (collapse the embed list and choose `BRWx::EIPManager` from the Local Objects list). Select the `ResetColumn` Method. Code placed here is executed each time a column is selected for editing. Insert the following code in this embed:

```
If Self.Column = 3
    SELF.EQ.Control.Feq{Prop:Xpos} = |
        SELF.EQ.Control.Feq{Prop:Xpos} - 10
END
```

This code first ensures that the edit field being created is the field you are interested in. In this case the checkbox is the third field in the browse. Next it takes the field equate for the EIP control just created and moves it 10 points to the left. This lines the entry checkbox up with the display checkboxes when in browse mode. `Self.Column` and `Self.EQ.Control.FEQ` are variables that are visible in the EIP class.

The only thing left to do is get rid of the ugly background and text. Unfortunately, you need to use another embed point to accomplish this. To understand why you can't use the same embed for all the code, you must understand what the EIP class is doing. During the `Init` method for the `EditCheckClass` for `AllowEntries`, the checkbox control is created. However, it is displayed in the class after the `init` code and before the code from Figure 7. This means that if you do not change the display characteristics they will be seen for a few seconds as the code is executed. However, the position of the control is not updated until the control is displayed. Therefore the code to move the control must exist in the `ResetColumn` embed.

To fix the background display problem, place two lines of code in the `Init` method for the `AllowEntries` EIP object as follows (look in the embed list under `LocalObjects|EditInPlace::jrn1:AllowEntries`):

```
Self.Feq{Prop:Text} = ''
Self.Feq{Prop:Trn} = True
```

These two lines remove the "@p p" from the prompt and make the background transparent. Your program will now display a checkbox in EIP mode that matches the box displayed in browse mode.

Summary

With the latest ABC templates TopSpeed has provided very powerful EIP abilities, and with just a little tweaking you can give your browses and data entry a much more polished look and feel.

[Download the source code](#)

[NextAge Consulting](#) is an independent consulting firm, based in Union MO, providing custom software and services, to small to medium-sized companies throughout the United States. NextAge focuses on client/server and distributed application development utilizing Clarion for Windows, and also produces Clarion Development Tools including the Imaging Template and the soon to be released Interface Templates. [Pete Halsted](#), the owner of NextAge Consulting, is a Clarion Certified Developer with 14 years of experience in the industry.

[ABC Embeds Are Easy](#)

(Dec 21, 1999)

[December 1999 News](#)

(Dec 21, 1999)

[Edit-In-Place CheckBoxes Done Right](#)

(Dec 21, 1999)

[A Class Wrapper For Files](#)

(Dec 21, 1999)

[Clarion Magazine Holiday Schedule](#)

(Dec 21, 1999)

Copyright © 1999-2000 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.

published by
CoveComm Inc.clarion magazine
Good help isn't that hard to find.**\$6.25/month**[Main Page](#)[Log In](#)
[Subscribe](#)[Frequently Asked
Questions](#)[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)[Advertising](#)[Contact Us](#)**Feature Article**

December, 1999

How Did Those Files Get Inside That Class?

by Jim Kane

While I'd dearly love to fill your heads with visions of assembler, registers, and COM, every now and again I have to write code for more mundane things like validating a product order and determining a projected delivery date. Naturally I'd like to get these things done fast and efficiently so I can get back to rearranging registers, stacks, and the API – who wouldn't?

The quickest way I can imagine to complete a task is to find that the code exists and can simply be plugged in. Nothing pleases me more that making a step by step list of how to complete a programming task only to find I have a pre-written class that covers each step. Those days do happen and are becoming more frequent as I expand my class libraries. Unfortunately, it's rare that something like an order entry system can be written from scratch. Usually there is a pre-existing file structure that must be followed. Since no one designs a file structure as well as (insert your own name here) the fields are never laid out quite right or named the correct way.

So how can you write the code once, yet fit as many file structures and field names as possible? The short answer is it can be done, in Clarion, with classes.

Inspiration

The task that inspired me to write the code this article is based on was one of those jobs that seemed simple at first. The company in question currently accepted orders from sales people who collected orders over the day and zipped them at night followed by an ftp upload. Other orders were taken over the phone, while still others came by fax. A different program took each order, depending on if it came via the sales force or from in-house (fax or phone).

Typically each order consisted of a main item with zero to 25 accessories. The order could not be shipped until all the accessories were available. Unfortunately the logic was fairly complex. If a customer ordered two types of input devices and only one was available, the order could be shipped. If they ordered only one input device and it was not available, the order could not be shipped.

My task was to write a web order entry system that calculated estimated shipping cost and date while the customer was still on-line. It sounded manageable at first, but when I entered

[ABC Embeds Are Easy](#)
(Dec 21, 1999)[December 1999 News](#)
(Dec 21, 1999)[Edit-In-Place CheckBoxes
Done Right](#)
(Dec 21, 1999)[A Class Wrapper For Files](#)
(Dec 21, 1999)[Clarion Magazine Holiday
Schedule](#)
(Dec 21, 1999)**TopSpeed****Clarion 5**
by TopSpeed**FREE** Microsoft
Internet
Explorer



some test orders the program that took orders from the sales force gave one answer for shipping date and shipping cost, and the program that took orders from telephone/fax gave a different answer. The file record structure and field names were also different for each order type. To add insult to injury, with the addition of a SQL based web order entry system a third new record structure was about to be introduced. This structure included a few new fields not needed by the other systems, such as an email confirmation address.

Due to some legacy reporting system requirements, file formats could not be merged. Also the discrepancy in the existing systems had to be resolved, as it was common to find calculated freight costs hundreds of dollars different from each other. I'd have a better chance of getting paid and coming out of this project alive if the calculated shipping cost was not only consistent, but accurate or higher than the real shipping cost!

The goal became to write one fairly complex piece of logic that took input from a minimum of three different file formats and wrote the projected ship date and estimated shipping cost back to the same order record the information came from. In spite of the fact that three different programs were involved, the results needed to be consistent. Since I was fairly confident I'd be learning business rules as I went along (read as: the customer didn't have a clue what he wanted but he'd sure be fast to tell me if I didn't deliver it!), I really wanted to only have to maintain the logic in one place.

The best way for me to do that was to write one encapsulated piece of code that could be called from each of the three programs and operate on different files. Nothing to it. All I had to do was write code that read and wrote files whose structure I did not know at the time I was writing the code.

Design Issues

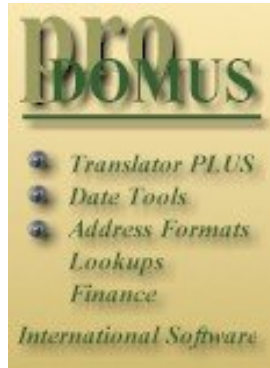
FileManager, one of the ABC classes, already provides a file independent method for many file operations, but what is missing is a way to address individual fields. ABC uses the field pairs class to copy from a file to a queue but that was not what I needed. I needed to be able the address fields like Product ID and Quantity Ordered as input regardless of the actual field name and write back the anticipated ship date and shipping cost.

For illustration purposes I'll define a few files and only deal with finding the estimated shipping date:

```
WebOrder :
Sysid
Estshipdate
EmailAddr

WebOrderDetail :
Sysid
WOSysid
OrderDate_Date
OrderDate_Time
ProductID
Qty
PhoneOrder
Sysid
Estdate

PhoneOrderDetail :
Uid
```



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

```

POSysid
OrderDate
ProductID
QtyOrdered

```

Inventory:

```

Sysid
ProductID
QtyOnHand
AvailDate

```

Both WebOrder and PhoneOrder are header files for an order. I will refer to them generically as Order. To the code that enforces the business rule, either WebOrder or PhoneOrder has a functional appearance of:

Order:

```

Sysid
EstShipDate

```

For each of the real files, WebOrder and PhoneOrder, the actual field names and data types may differ. Each file may contain additional fields not logically important. Using the same thought process I developed a generic Detail file to virtualize WebOrderDetail and PhoneOrderDetail:

Detail:

```

ParentSysid
ProductID
QtyOrdered

```

The business rule which needs to be enforced is written in pseudo code as:

```

! set() on primary key of order file
Set(keyOnOrderSysid)
Loop
  Next(Order); exit on error
  Clear(EstShipDate)
  LastParent = Sysid !save the linking field
  !Set up child key fields
  Set(DetailKey)
  Loop
    Next(Detail)
    If Detail.ParentSysid<>LastDetailParent or error
      Order.EstShipDate=EstShipDate !Save the ship date
      Update(Order)
      Break
    End
    Inventory.ProductID = Order.ProductID
    ! Fetch gets QtyOnHand and AvailDate
    Fetch(KeyOnProductID in Inventory file)
    If QtyOnHand<=QtyOrdered and AvailDate>EstShipDate
      EstShipDate = AvailDate.
    End
  End !loop Detail - child
End !loop Order - parent

```

The requirement is to be able to do Set, Next, Fetch, and Update on files. As well, the code must get and set a field's value in spite of the fact that the fields are unknown at the time of writing. All that is known is a generic, virtualized version of the files involved: Order, Detail, and Inventory. The real files may have more fields and the field names may vary.

Since the main work to be done is to manage access to unknown fields, I hereby christen the new class that will solve the problem `FldMgrCl`, short for Field Manager Class. At its heart it will have to manage a list of fields (`FieldListQ`) and keys (`KeyQ`) plus have the ability to read and write those fields. Since the data type of the fields is not know, the Clarion Any type sounds like the approach to take. In the `Init` and `Kill` methods, the primary task will be to create and destroy the `FieldListQ` and `KeyQ`. In addition the class will reuse ABC file manager methods so the class will have to store references to these objects. Putting these bon mots (said with a New York accent if you know me) into code what you have so far is:

```
!ABCIncludeFile
OMIT('_EndOfInclude_',_FldMgrClPresent_)
_FldMgrClPresent_ EQUATE(1)
!Other Classes
    Include('ABFILE.INC')
!Equates - will be seen by using program
FieldListQtype Queue,Type
FieldName      string(20)
FieldRef       Any
    End
KeyQtype       Queue,type
KeyName        string(80)
TheKey         &key
    end

FldMgrClType Class,type,module('FldMgrCl.CLW'),
    LINK('FldMgrCl.CLW',_ABCLinkMode_),DLL(_ABCDllMode_)
!Member Data
FieldListQ     &FieldListQType    !List of fields
KeyQ           &KeyQType           !List of Keys
File           &File               !ref to file
Buffer         &Group              !ref to buffer
FM             &FileManager        !ref to FileManager
RM             &RelationManager    !ref to RelationManager
!Set up Methods - Calls are template generated
Init           Procedure()
Kill           Procedure()
AddFile        Procedure(File pFile, *Group pBuffer)
AddManagers    Procedure(FileManager pFM, RelationManager pRM)
```

Notice how `ABFile.inc` is simply included in the new class. With that done `FileManager` and `RelationManager` can be referenced and used freely. Not bad for one line of code! OOP is plug and play; this is but one example of how classes can be combined to make something powerful.

Before I move on I need to comment on the code in `Kill`, as the ANY data type requires some special care and feeding or it *will* bite you (and not in a good or playful way!).

```

FldMgrClType.Kill                Procedure( )
!destroy any dynamic objects and clean up
Recs long,auto                  !junk queue record count
I    long,auto                  !junk index variable
Code
!Null the ANYs in the FieldHdrList
!then then dispose of the Q
If ~SELF.FieldListQ &= NULL Then
    Recs = Records(SELF.FieldListQ)
    Loop I = 1 to Recs
        Get(SELF.FieldListQ,I)
        !ANY field set to null
        SELF.FieldListQ.FieldRef &= NULL
        Put(SELF.FieldListQ)
    End
    Dispose(SELF.FieldListQ)
End
If ~SELF.KeyQ &= NULL then Dispose(SELF.KeyQ).
RETURN

```

Notice where the ANY field in the queue, `FieldRef`, gets set to `NULL`. This allows the ANY variable a chance to clean up and release its memory. Also note that since in `Init` the queues were `NEW`(ed), in `Kill` they are `DISPOSE`(ed). By cleaning up the ANY's and `DISPOSE`(ing) all that is `NEW`(ed), memory leaks are avoided. If you create memory leaks by forgetting these things, it's quite possible for the program to run out of memory after a while resulting in all kinds of hard-to-trace errors. Since I have brief moments of sanity, I try hard to use them to match all memory allocating activities with memory freeing activities and avoid the problems.

At this point I suggest downloading and opening the [sample code](#). In particular look at `fldmgrcl.clw` and `fldmgrcl.inc` for the `Init` and `Kill` code, plus the rather mundane code to store away the file, buffer, `FileManager` and `RelationManager`.

I suppose it's a good thing I like to program. I do not think I would make it in advertising, since I just told the reader (that's you!) to download and open something, then warned that is only mundane. Nonetheless, read the code.

Now that `FieldListQ` and `KeyQ` have been created, the logical thing, as Mr. Spock would say, to do is fill them:

```

AddKey           Procedure(string pKeyName, Key pKey)
AddField        Procedure(String pFieldName, *? pField)

```

Each of these methods takes the name of a field or key and the field or key itself so it can store a reference. For those for whom references are unfamiliar, think of them as pointers, and a way for a computer to remember where a field or key lives. A typical call to these methods for the order file would be:

```

WebOrderFieldMgr.AddKey('WEB:BY_SYSID', WEB:BY_SYSID)
WebOrderFieldMgr.AddField('SYSID',SYSID)

```

Now inside the `WebOrderFieldMgr` class the `Web:By_Sysid` key or the `Sysid` field can be referred to by the strings `'Web:By_sysid'` and `'Sysid'`. I could have chosen to

use equates rather than strings which probably would have been faster. I chose not to because for a large dictionary and many files, that would create a large number of labels. In one case this brought back some pool limits I'd rather not see again.

In my implementation, the strings are not case sensitive. The code for adding the key or field to its respective queue is straightforward. Perhaps more interesting is the code to get or set the field values:

```

Add a field:
FldMgrClType.AddField      Procedure(String pFieldName, ←
                          *? pField)

  code
  Assert(~SELF.FieldListQ&=NULL)
  Clear(SELF.FieldListQ)
  !Save the reference
  SELF.FieldListQ.FieldRef &= pField
  !save the token to identify the field
  SELF.FieldListQ.FieldName = Upper(pFieldName)
  ADD(SELF.FieldListQ,SELF.FieldListQ.FieldName)
  Return

Save or set a field value:
FldMgrclType.SetField      Procedure(String pFieldName, ←
                          ? pTheValue)

  Code
  SELF.FieldListQ.FieldName = Upper(pFieldName)
  Get(SELF.FieldListQ,SELF.FieldListQ.FieldName)
  If ~ErrorCode() then
    !Save the value passed in the actual
    ! field referenced by fieldref
    SELF.FieldListQ.FieldRef = pTheValue
  else
    If ErrorCode()
      message('Error locating field:&clip(pfieldName)|
              &' Error:&Clip(Error()),'SetFieldError')
    end
  end
  Return

Return or get a field value:
FldMgrClType.GetField      Procedure(String pFieldName)
  code
  Assert(~SELF.FieldListQ&=NULL)
  SELF.FieldListQ.FieldName = Upper(pFieldName)
  Get(SELF.FieldListQ,SELF.FieldListQ.FieldName)
  Assert(ErrorCode()=0)
  Return SELF.FieldListQ.FieldRef

```

Notice in the prototypes the use of the ? and *? data types. These can be thought of as parameters able to pass any data type. They're ideal for the current work at hand. For those not use to working with references notice two different syntaxes for two different purposes:

```
FieldRef &= PRE:FIELD
```

means that `FieldRef` now points to a field called `PRE:FIELD`, while

```
FieldRef = 12
```

means the value 12 should be stored in what ever `FieldRef` points to (`PRE:FIELD` in this case). This can be confusing because the ANY data type can be used either way.

The net result of all this is the ability to refer to a field or key by an equivalent string token. You can even alias or provide an alternative string token that can then be used to reference a field. For example:

```
FldMgrClType.AliasField Procedure(String pFieldName, ←
                                String pAliasName)
TempAny Any
  Code
SELF.FieldListQ.FieldName = Upper(pFieldName)
  Get(SELF.FieldListQ,SELF.FieldListQ.FieldName)
  Assert(ErrorCode( )=0)
TempAny &= SELF.FieldListQ.FieldRef
!must clear a queue containing
!any an ANY before reusing the Any
Clear(SELF.FieldListQ)
!Set up and save two fields - An Alias is born
SELF.FieldListQ.FieldRef &= TempAny
SELF.FieldListQ.FieldName = Upper(pAliasName)
ADD(SELF.FieldListQ,SELF.FieldListQ.FieldName)
  Return
```

Now either the `AliasName` or the `FieldName` string can be used to reference the field. The above code takes the field name passed and locates it in the `FieldListQ`. If it is not found, usually a typing error, an `Assert ()` fires. Once the `FieldListQ` entry is found, the reference to the "real" field is saved in a temporary variable (`TempAny`). After clearing the queue buffer (required when the ANY data type is used in a queue), the `AliasName` and reference to the real field is stored in the queue.

On subsequent accesses, whether the lookup into the queue is done using the fieldname or the alias name, the same field reference is found. This allows fields with different names in different files that have the same purpose to be referred to in code by the same name. For example, consider two files `WebOrder` with a field `Qty` and `PhoneOrder` with a field `QtyBought`. Either `WebOrderFldMgrCl` or `PhoneOrderFldMgrCl` could be passed as an input parameter to an `OrderProcessing Class`. Use of aliases makes it easy to pass either `WebOrderFldMgr` or `PhoneOrderFldMgr` even though the field names differ and get the job done. For example:

```

!Set up a QuantityOrdered Alias in both files
Weborderfldmgr.AliasField('Qty','QuantityOrdered')
PhoneOrderFldMgr.AliasField('QtyBought','QuantityOrdered')

!Pass either fieldManager class to the OrderProcessClass:
OrderProcessClass.AddOrderFile(WeborderFldMgr)
!or
orderProcessClass.AddOrderFile(PhoneOrderFldMgr)

!Code in Order ProcessClass
OrderProcessClassType.AddOrderFile(FldMgrClType OrderFile)

Code
SELF.OrderFile &= OrderFile
!Inside OrderProcessClass the totals
! are calculated the same:
LineItemCost = |
    SELF.OrderFile.GetField('QuantityOrdered') * UnitCost
!Note: I don't care if this is WebOrder or
! Phone Order! Both have the same alias

```

Time for a test! I said near the start the idea was to encapsulate the business rule (three points for Kane for successfully using a buzz word/phrase). The first step is to create FieldManager classes for the WebOrder, WebOrderDetail, PhoneOrder, PhoneOrderDetail, and Inventory file. Since typing all the field names resembles work, its time for a template. The accompanying code contains a template to generate a FldMgrClass wrapper for each of the files. Thanks to the use of the Alias method above, all the Detail files (WebDetail and PhoneDetail) now have a ParentSysid, ProductID field. For example:

```

PhoneOrderDetailFldMgr.AddField('POSysid', PDet:POSysid)
PhoneOrderDetailFldMgr.AliasField('POSysid','ParentSysid')
WebOrderDetailFldMgr.AddField('WOSysid', WDet:WOSysid)
WebOrderDetailFlgMgr.AliasField('WOSysid','ParentSysid')

```

Now both files have a field that can be referred to by the string 'ParentSysid'. Notice also that the template generated the bulk of the code. All that needed to be typed were some aliases. No carpal tunnel syndrome here!

Time to put it to use:

```

ProcessOrderClType Class,type,module('FldMgrCl.CLW'),
    LINK('FldMgrCl.CLW',_ABCLinkMode_),DLL(_ABCDllMode_)
OrderFile &FldMgrClType
DetailFile &FldMgrClType
InventoryFile &FldMgrClType
Init Procedure(FldMgrClType pOrderFile,
    FldMgrClType pDetailFile, FldMgrClType pInventoryFile)
Kill Procedure()
DoIt Procedure()
End

```

Using SELF.OrderFile.GetField('ParentSysid') you can obtain the value of

either WOSysid or POSysid depending on which file was passed in the DetailFile parameter. In other words, after setting up all the field manager classes both the WebOrder and PhoneOrder files can be processed with the same code in the DoIt method of the ProcessOrderCl.

```

ProcessOrderCl ProcessOrderClType
Code
!set up field managers here
!add in aliases for ProcessOrderCl
ProcessOrderCl.Init(WebOrderFldMgrCl, ←
    WebOrderDetailFldMgrCl, InventoryFldMgrCl)
ProcessOrderCl.DoIt
ProcessOrderCl.Kill
ProcessOrderCl.Init(PhoneOrderFldMgrCl, ←
    PhoneOrderDetailFldMgrCl, InventoryFldMgrCl)
ProcessOrderCl.DoIt
ProcessOrderCl.Kill

```

Success! The code is using files inside a class and small changes in field names or file formats just don't matter: they can still be processed. Consider the possibilities in writing generic code that works on version 1, version 2, and version 3 file formats for a program that has evolved over the years.

At the start I said I wasn't going to *fill* your head with COM or other low level stuff. Notice the word *fill* is in italics so that means I can still legally at least mention COM.

Consider the scenario I outlined through out the article of the order entry system. To update each program I need to insert the ProcessOrderCl, and recompile if the process of assigning an estimated shipping date changes. In the real world the code is much more complex and in practice changes at least once per month. On the other hand, if I could write the code as a COM object I could replace that object at any time. The next time any of the programs that use the COM object ran, the new object would be loaded into memory and executed. No recompile.

For connected uses, I could even employ DCOM and only update the COM object on one machine and any one across the enterprise would have an instant update the next time they ran a program that needed this COM object. Not only that but because the COM objects are universal, the same COM object could be used in programs written in any COM aware language, including Active Server Page scripts.

While in this case the code is only going into three or so programs and recompiling and redistributing those programs is not a large chore, consider a more common piece of code or business rule like perhaps a discounting scheme that gets used in Order Entry, Accounting, Sales Projections on and on through out the enterprise. It would not be surprising for some rules to touch many, many programs. By encapsulating the rule in a COM object, I could hand out updates rather easily. Very attractive. Microsoft has a name for this concept; it's called DNA, or Distributed interNet Applications. Of course, putting this into practice requires a development tool that can write COM objects.

Maybe some day Top Speed will see the light. Until that time, at least this code comes one step closer, in that it is only necessary to recompile to take advantage of updated order processing code without rewriting the code for three different file structures.

[Download the source code](#)

[Jim Kane](#) was not born anywhere 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.

[ABC Embeds Are Easy](#)

(Dec 21, 1999)

[December 1999 News](#)

(Dec 21, 1999)

[Edit-In-Place CheckBoxes Done Right](#)

(Dec 21, 1999)

[A Class Wrapper For Files](#)

(Dec 21, 1999)

[Clarion Magazine Holiday Schedule](#)

(Dec 21, 1999)

Copyright © 1999-2000 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

Clarion
Development
Resources

published by
CoveComm Inc.

clarion magazine
Good help isn't that hard to find.

\$6.²⁵/month

[Main Page](#)

[Log In](#)
[Subscribe](#)

[Frequently Asked
Questions](#)

[Site Index](#)
[Article Index](#)
[Author Index](#)
[Links To
Other Sites](#)

[Downloads](#)
[Open Source
Project](#)
[Issues in
PDF Format](#)
[Free Software](#)

[Advertising](#)

[Contact Us](#)

Press Release

December, 1999

Clarion Magazine Christmas Schedule

December 21, 1999

The Clarion Magazine office will be closed from Tuesday, December 21, through Sunday, January 2, 2000. Clarion Magazine will next publish on January 4, 2000.

Automated orders will be processed as usual during this time, meaning that if you're a new subscriber you'll get access within minutes of placing your order (unless you happen to choose a user id also used by an existing subscriber).

A Merry Christmas and a Happy New Year to all of you from Clarion Magazine!

Dave Harms

Copyright © 1999-2000 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.

[ABC Embeds Are Easy](#)
(Dec 21, 1999)

[December 1999 News](#)
(Dec 21, 1999)

[Edit-In-Place CheckBoxes
Done Right](#)
(Dec 21, 1999)

[A Class Wrapper For Files](#)
(Dec 21, 1999)

[Clarion Magazine Holiday
Schedule](#)
(Dec 21, 1999)

TopSpeed

Clarion 5
by TopSpeed

FREE Microsoft
Internet
Explorer