



[Home](#)

[Subscribe](#)

[E-Books](#)

[News](#)

[Blog](#)

[Store](#)

[My ClarionMag](#)

[My Lists](#)

[Contact](#)

Clarion Magazine

This edition includes all articles, news items and blog posts from June 1 2010 to June 30 2010.

Clarion News

[Read 20 Clarion news items.](#)

The ClarionMag Blog

[Read 8 blog entries.](#)

Articles

[Queues in Dictionaries](#)

June 2 2010

Queues in the dictionary can be very handy. Dr. Parker shows how it's done.

[Clarion 7.2 Released](#)

June 3 2010

SoftVelocity has released Clarion 7.2 to CSP program participants. Third party component DLLs must be rebuilt with the Clarion 7.2 compiler and runtime. As well, the .APP file format has been updated in this release and .App files opened in 7.2 cannot be opened with prior versions.

[Welcome To The New ClarionMag Site!](#)

June 4 2010

Welcome to the new site! Here's a summary of some of the new features. Look for more detailed explanations in the days and weeks to come.

[Editable Dictionary Queues](#)

June 15 2010

In the previous article in this series, Steve Parker showed how to declare queues in the dictionary. Now it's time to complete the picture and show how to make a fully functioning browse and form to maintain a queue.

Everybody Needs Some Closure

June 15 2010

Closures are a hot topic in programming circles, thanks to the growing influence of functional programming. But do you really need to understand closures? Dennis Evans thinks you do. In this article he covers the concepts, shows how to write a closure in Clarion#, and explains the mysteries of the underlying IL code.

ClarionMag's Subscription Rate Increase Coming July 18

June 16 2010

Our rates will be going up on July 18, 2010. Subscribe or renew now and save!

Making a Splash

June 24 2010

When you have large applications, especially applications with lots of DLLs, it can take quite a bit of time for the program to load and for the frame screen to appear. Users don't like long periods with no response, so this effect is not ideal. As Bruce Johnson explains, there are ways to speed up application load time, but often the best solution is a splash screen.

Queue Record Structures

June 29 2010

As Steve Parker recently explained, it's possible to use a queue declared in the dictionary to create a fully editable browse/form. But is it possible to go one step further and treat a queue just as you would a file's RECORD structure?

Clarion 7.2 Makes Steady Progress

June 30 2010

On June 3 2010 SoftVelocity released Clarion 7.2 build 7232, followed a week or so later by build 7248. Dave Harms takes a look at both builds.

Queue Browsers, Sort Headers and Locators

June 30 2010

There are two final functionalities Steve Parker needs to implement to make his queue-based list and form completely mirror the browse-forms options provided by the templates: sort headers and locators.

Copyright © 1999-2010 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.

Clarion News

[iQ-XML 2.70](#)

A new update for iQ-XML (version 2.70) is now available for both Clarion 6.3 and Clarion 7.1. Features include: Added optional Date/Time parameters to the ConvertToGMT function; Added XML:SetEncoding to force encoding type (Parser); Fix - Field Pictures may not work when adding from Group or Queue; Fix - the AutoCloseParent in the AddFromGroup would not work when forced as False rather than left blank. The default value for AutoCloseParent is TRUE unless specified.

Posted June 9 2010 ([permanent link](#))

[SendTo Goes Light](#)

SendTo basically takes your browse/List and exports it (Sends it) to a number of outputs (printer, CSV, Excel, Word, XML, email, FTP, clipboard, PDF). In time for SendTo's sixth birthday, the requirements for the other tools (OfficeInside, NetTalk, Winevent and Draw) has been removed from the templates and source. If you don't have all the added tools you can get going with the SendTo Printer, SendTo htm and SendTo csv. If you add OfficeInside in to the equation, you get SendTo Excel and SendTo Word. If you then add NetTalk, you get SendTo Email and SendTo FTP. So if you want you can "go to town" and get the whole bang shoot. Or if you're finding it a hefty price tag, you can get the shell, and add the functionality in, as and when you can. You can leave the SendTo Doc and Excel temptingly disabled for your clients to salivate over, and charge them \$500 for the "feature". The new price for SendTo is \$247.

Posted June 9 2010 ([permanent link](#))

[iQ-Sync 1.20](#)

iQ-Sync version 1.20 is now released. There is an enhancement in the way "in-use files" are checked, and a number of fixes. iQ-Sync is a free utility which allows you to create backup sets and copy projects, and run them quickly and easily. You can also add actions at the beginning and end of each backup set such as copy, delete, rename and even FTP.

Posted June 9 2010 ([permanent link](#))

[CommandBars Wrapper Template 2.05](#)

The CommandBars Wrapper Template version 2.05 is now available for download. This is a

small update but includes a very nice feature that was mentioned by Paul Blais during the last ClarionLive Noyantis user group meeting. Basically, the template now generates an "accepted" embed point for each control that is defined within the template. This makes it much easier to add any embed code to a specific CommandBar control. The generation of these types of embed points will be added to ALL of the other Noyantis Wrapper templates in the future. Also, a new method called 'GetDispatch' has been added which comes in handy when linking the CommandBars control to other controls such as DockingPanels etc. The new version can be downloaded using the link specified in your original sales email.

Posted June 9 2010 ([permanent link](#))

NetTalk WebShop South Africa Early Bird Discount

Capesoft's NetTalk WebShop basically a workshop, except you don't do any work, you just have fun writing web apps. You bring your laptop, and a web-app (or two or seven) that you're busy with, and come spend two days with the Capesoft team to enhance them, and take them to a new level. The important thing to note is that bookings are open for the South African and Australian legs. There's an early bird discount for those who book and pay before the Soccer World Cup (SWC) kicks off on 11 June. The normal price for the Webshop will be \$550 for both days, but you can grab your slot for \$375 (for Cape Town or Jo'burg) and \$400 (for Sydney or Melbourne) if you book and pay on or before the 11 June 2010. If you leave it to the kick-off between Mexico and South Africa, you'll have left it too late to grab a shot at the early bird discount. Snooze you lose.

Posted June 9 2010 ([permanent link](#))

NetTalk WebShop Australia Early Bird Discount

Capesoft's NetTalk WebShop basically a workshop, except you don't do any work, you just have fun writing web apps. You bring your laptop, and a web-app (or two or seven) that you're busy with, and come spend two days with the Capesoft team to enhance them, and take them to a new level. The important thing to note is that bookings are open for the South African and Australian legs. There's an early bird discount for those who book and pay before the Soccer World Cup (SWC) kicks off on 11 June. The normal price for the Webshop will be \$550 for both days, but you can grab your slot for \$375 (for Cape Town or Jo'burg) and \$400 (for Sydney or Melbourne) if you book and pay on or before the 11 June 2010. If you leave it to the kick-off between Mexico and South Africa, you'll have left it too late to grab a shot at the early bird discount. Snooze you lose. Note to Australian attendees - please rather book at the Melbourne leg (if you need to travel anyway), as numbers in Sydney are pretty tight.

Posted June 9 2010 ([permanent link](#))

Laro Group Templates C7.2 Compatible

All the Laro Group template products install and register properly with Clarion 7.2. The installers still find and identify Clarion 7.2 as if it were Clarion 7.1, but if you select it for the

installation everything installs properly in Clarion 7.2. As these products are all source code and don't include any DLLs, there are no problems with the change of version.

Posted June 9 2010 ([permanent link](#))

PlugIT Clarion 7 Compatible

PlugIT is now Clarion 7 compatible. This is a free upgrade for registered users.

Posted June 9 2010 ([permanent link](#))

AFE For C7.2

AFE for C7.2 is available for download.

Posted June 9 2010 ([permanent link](#))

Xtreme ToolkitPro, Xtreme SuitePro 13.4.0

This is a maintenance and support release that addresses some outstanding issues that were not addressed with 13.3.1 including feature enhancements and fixes.

Posted June 9 2010 ([permanent link](#))

RPM Build "G" For 6.3.9051+

RPM build "G" for 6.3.9051+ is available. In addition to the RPM Dummy Report fix and improved Office Inside support this build has better user option and file support. All file folder and name variable lengths have been extended, in some cases significantly. INC files are now used for the option group TYPE definitions and centralize INI/Registry functions have been added to the "B" source module. The "Release Date" mirrors the previous C7 release for download access. In process is the back port to legacy and added support for Fomin, ReportDAT! and ReportWriter 7. The next major addition to RPM is SQL report archiving which is already in design.

Posted June 9 2010 ([permanent link](#))

iQ-XML Clarion 7.2

iQ-XML is now available for Clarion 7.2 iQ-XML is a free tool for Clarion developers to add XML functionality to their applications with very little knowledge. It offers many features not found in Clarion's own XML functions. iQ-XML comes with both Parser and Writer functions. Generate an XML document from a Clarion Queue, Group structure, or using the APIs.

Posted June 9 2010 ([permanent link](#))

amazingGUI 1.2.3.0

amazingGUI 1.2.3.0 has been released. New in this release: Clarion 7.2 installer included; New method `agWindowManager.AddComponent` has been made public in order to enable runtime created controls to be added to amazingGUI; A new sample project has been added showing how to apply amazingGUI to runtime created controls.

Posted June 9 2010 ([permanent link](#))

J-Html and J-Spell Clarion 7.2 Compatible

Strategy Online has released Clarion 7.2 compatible builds of J-Html and J-Spell.

Posted June 9 2010 ([permanent link](#))

ClarionTools 7.2 Compatible

ClarionTools has announced the immediate availability of the Clarion 7.2 compatible installers, DLLs, and LIBs for the entire product line.

Posted June 9 2010 ([permanent link](#))

Super Stuff Update

There's a new and improved version of Super Stuff (MH), with improvements in mhView, Threaded Browse-Forms, and Browse Active Filter. There are also new DropButtons for modern-looking drop lists. The following are all fully tested and compatible with Clarion 6.x through 7.2: Super Browse; Super Import-Export; Super Invoice; Super QBE; Super Security; Super Stuff (MH); Super Tagging. The planned order for the remainder is: Super Passcode; Super Limiter; Super Field-Filler; Super QuickBooks-Export. These will be released over the coming weeks.

Posted June 9 2010 ([permanent link](#))

CHT Build 14B1.01

CHT Build 14B1.01 has been posted. The CHT toolkit doesn't use DLLs compiled with Clarion so there are no compatibility issues. However, the pre-update CHT template chain makes a specific check for C7.1 and may stop you using it with C7.2. The updated build just posted will get you up to speed with C7.2 and is okay for use with C7.1 as well. The demo apps with this build update have been converted to C7.2 format, so you will not be able to open them up in C7.1. The 14B1.01 (June 7,2010) build templates, classes, DLLs and utility executables present no compatibility issue whether you're using C7.1 or C7.2

Posted June 9 2010 ([permanent link](#))

Noyantis Templates Clarion 7.2 Compatible

New versions of all Noyantis templates are available. The installers are now Clarion 7.2 compatible. Also the Codejock wrapper templates now support v13.4.0.

Posted June 9 2010 ([permanent link](#))

iQ-Sync 1.22

iQ-Sync 1.22 has been released. Some minor changes, compiled in C7.2 and the enhancement to now optionally run the copy after a preview.

Posted June 9 2010 ([permanent link](#))

Comsoft7 Installers Updated For 7.2 Detection

Comsoft7 has updated 14 product installers to detect Clarion 7.2.

Posted June 9 2010 ([permanent link](#))

Copyright © 1999-2010 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.

The ClarionMag Blog

Search problem fixed

I introduced a very silly bug last night when I posted an update, with the result that titles disappeared from the search. A temp on-site fix broke article viewing, so I've uploaded a proper fix. Everything should be cool again.

There may be a few brief interruptions today as I integrate improved error handling on the site.

Posted June 1 2010 ([permanent link](#))

ELMAH rocks!

I've added [ELMAH](#) to the ClarionMag site - this is an open source error logging product for ASP.NET MVC (and regular ASP.NET) that traps and reports on page errors.

The very first thing ELMAH told me was that I didn't have favicon.ico set up. Yikes!

The second was that there's an issue with paging on the news page, but only in conjunction with the URL rewriting for the old news page. Backward compatibility can be a bear. But that one's fixed now.

I'm sure there will be other issues. If you get an error page telling you the error has been logged, that means it is on my radar.

Posted June 1 2010 ([permanent link](#))

News navigation

[ELMAH](#) notified me of a problem with the news page navigation. That was a regression - fixed, apologies for the error.

I'm also seeing some issues with invalid characters in search strings, but I haven't posted an update for that yet.

On a side note, ELMAH has also notified me of a number of script kiddie hack tries. They're not going to work, especially when they're attempts to execute PHP files. But if you have an ASP.NET (classic or MVC) web site, I strongly encourage you to install ELMAH. You'll probably be surprised to see what's really happening.

Posted June 4 2010 ([permanent link](#))

Reader comments are back!

You can once again post reader comments! Also you can view the most recent comments [here](#). I'll put up a link on the home page shortly, and will also set up a redirect from the old comments page.

Posted June 9 2010 ([permanent link](#))

Notes on reader comments and article searching

A few points to keep in mind when posting reader comments:

- You need a blank line between paragraphs of text
- You can tag text as source code
- You need to add a space after any < character or it'll be taken as a tag. And if it's not one of the allowed tags (e.g. for bold, italic, code etc) the text will be stripped out.

- Be sure to look at the preview to be sure your message is displaying as expected.

I've also made some changes to the search page. You can now search for text containing colons, such as [prop:sql](#). Prior to this change that search term would result in an error, and you had to use the term "prop sql" instead. That trick is no longer necessary.

Posted June 10 2010 ([permanent link](#))

Working on PDFs

As you may have gathered if you're following ClarionMag's Twitter feed (look to the right) I'm revamping the mag's PDFs, and I'm attempting to do this with WebSuperGoo's [ABCpdf](#) product.

ABCpdf is pretty cool, and does a much, much nicer job on the PDFs than Acrobat ever did. The only issue I'm having is that not all the links are being resolved within the document. Must be some bonehead mistake on my part, but I can't see what it is. I've emailed the company, so we'll see what comes of that.

Posted June 25 2010 ([permanent link](#))

New PDF format

The April/May PDF is now available for [download](#). In the past we created PDFs using Adobe Acrobat, with less than stellar results. For the new site we've implemented PDF creation using [webSupergoo's ABCpdf](#), which oddly enough is *not* a Clarion-related product. It's a .NET product, and we chose it because we needed something that would work with ASP.NET MVC.

The upshot is that our PDFs now suck a whole lot less. There's still some work to be done, but if you'd like to see the new look just download the [April/May PDF](#) (subscription required).

Posted June 29 2010 ([permanent link](#))

Free articles index is back

The free articles index can now be found [here](#). These are articles that are available without charge; some require a logon.

Posted June 30 2010 ([permanent link](#))

Copyright © 1999-2010 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.

Queues in Dictionaries

By Steven Parker

Posted June 2 2010

Two things I've really wanted are the ability to declare:

- queues in the dictionary

and

- files/tables in the Local and Module data pads

In a news group thread concerning an issue with 7.1.6955, Anthony Robinson provided an example of how to create a queue in the dictionary. Then Russ Eggen reminded me that this facility has been available since Clarion 6.

A little research showed that I had, indeed, created queues in the dictionary in C6. Having forgotten, it's time I wrote down the steps. Then, the next time I forget, someone can remind me, as Bruce Wojick (C3 Development, for those of you who have been around a while) once laughingly did when I complained of not being able to do something: "You just wrote the article on doing that!"

Queues in the dictionary

Clarion 6 introduced the ability to create global variables in a dictionary. In Clarion 6, when adding a new table, tick "Global" in the "Usage" option structure:

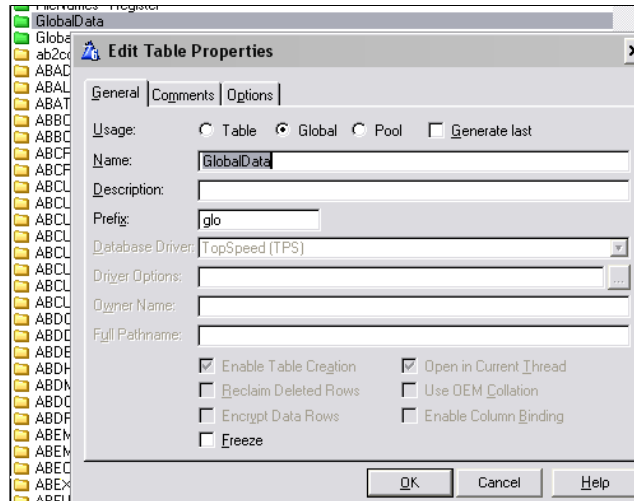


Figure 1. Creating a dictionary global data section in Clarion 6

In Clarion 7, there is an additional window (actually, a drop down) to deal with because the options have been relocated. First, select “Globals” from the menu on the left. Then select “Add Global:”

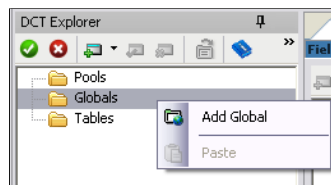


Figure 2. Creating a new dictionary global data section in Clarion 7

Next, provide a LABEL for the new data structure:

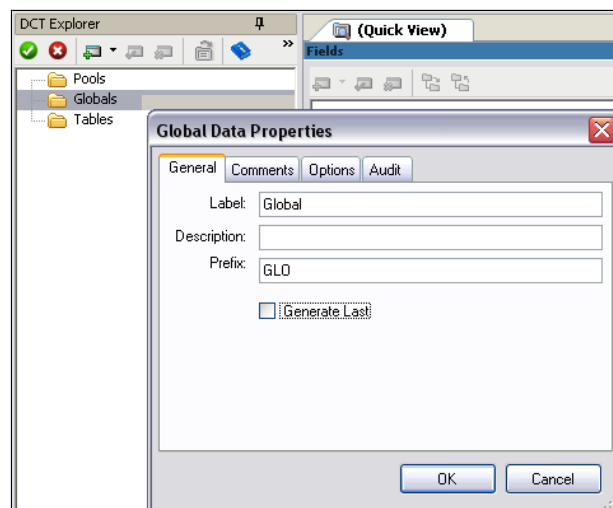


Figure 3. Label the global data structure

Do *not* forget the Prefix prompt (the Dictionary Editor won't let you leave it blank).

Now I can declare a variable. “Queue” is one of the options available and, therefore, by

selecting it, I create a queue within my global data section:

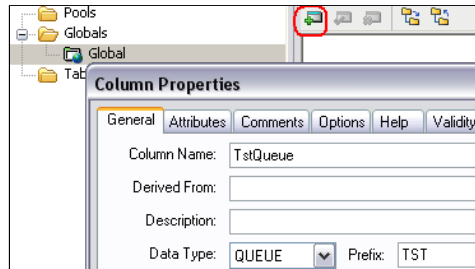


Figure 4. Creating the queue

Again, complete the Prefix prompt. I can now begin inserting fields into the queue definition:

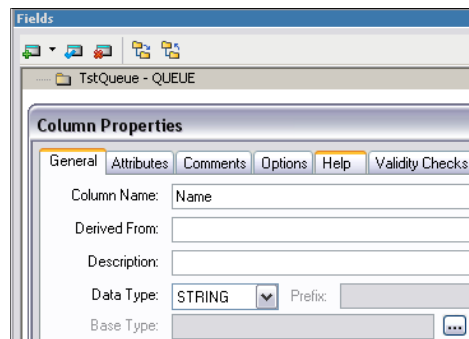


Figure 5. Inserting a field into the queue structure

As each field is entered, Clarion's default behavior is to assume I want to create another new queue field. This has been Clarion standard Dictionary Editor behavior for a while. When I'm done with my queue declaration, I hit "Cancel" on the Column Properties dialog.

Note: the queue will inherit the prefix of the global data section just created. The queue's elements will use the prefix of the queue.

If I want another global queue, I can create another. But, because I have just been working with a queue, the IDE asks me whether I want to add my new field after the queue or in the queue, beginning or end:

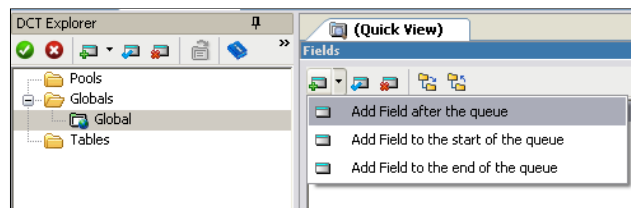


Figure 6. Creating a new field after having created a queue

Having, in the past, added fields to a queue that I intended to be outside the queue (and *vice versa*), this is a nice new feature. Essential? No. But nice to have.

If I click “Add Field after the queue,” I get a new Column Properties dialog and can create another queue (or other free-standing variable):

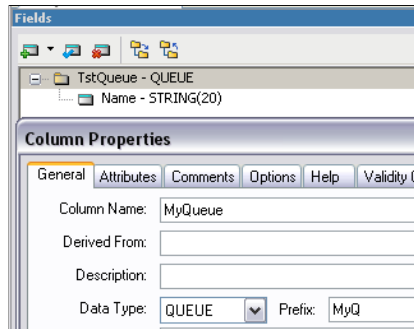


Figure 7. Creating a second global queue

When I complete this second queue, my dictionary looks like this:

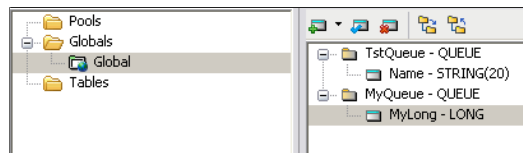


Figure 8. Two global queues in the dictionary

The sample dictionary, Queue.dct, which you can download at the end of this article, is the dictionary Tony Robinson provided with a second global queue added. In the version of this DCT provided, I added the second queue using a second global data section (see Figures 1 and 2). I am also including Tutor.DCT for Clarion 6 (from the Examples subdirectory) showing a global queue in a Clarion 6 dictionary.

If you are not inclined to commit memory at Dictionary Initialization, the TYPE attribute is available in both Clarion 6 and 7 on the queue’s Column Properties dialog. Of course, if I use the TYPE attribute, I need to remember to create an actual queue in my code.

Why Use Dictionary Queues?

For any file/table or global variable declared in the dictionary, Clarion’s templates ensure that the declarations in all .APPs are correctly handled. In a multi-DLL app, the data DLL correctly exports the declarations. All other .APPs are correctly coded, for me, with the EXTERNAL and DLL (when required) attributes. (This feature is *only* relevant for multi-DLL apps – however, planning against the day I need to split an app into multiple DLLs, I think it advisable to declare as much as possible in the dictionary.)

For example, if I create a global variable, AutoCl eanOnSal e, in a global data section with the prefix glo:, Clarion 6 generates:

```
glo: AutoCl eanOnSal e  BYTE(1)
```

in the source code for the data DLL and:

global: AutoCleanOnSale BYTE(1), EXTERNAL

for all other DLLs using this dictionary. In other words, putting things in the dictionary reduces coding and, more important, reduces coding errors.

By itself, this is sufficient to justify this exercise. Global queues, like any other global variables, become multi-DLL friendly.

Also, anything declared in the dictionary is visible to the Report and Window formatters. Similarly, variables declared in the C7 Data Pads or through C6's various data buttons are visible to the formatters in the relevant scope. That is what distinguishes such variables from variables declared in data embeds.

And, for me, *that* is the point of the exercise. If my queue is visible to the window formatter, I can create a browse on the queue. That is something I often want to do. (Of course, all the usual cautions against indiscriminate use of global data apply.)

Browsing a Queue

To browse a queue I start with a Window procedure, from the Templates tab.

On this window, I want to populate a browse box. However, I do not want to use the BrowseBox from the Control Templates pad.

That control assumes I will be using a file. It will not allow me to select my queue. (In Clarion 6, this is the equivalent of populating by selecting Populate | Control Template and selecting "Browse Box.")

Instead, use the Clarion Window Controls pad. (This is the equivalent of Clarion 6's Control | Listbox and, when prompted, selecting "Populate control without control template.")

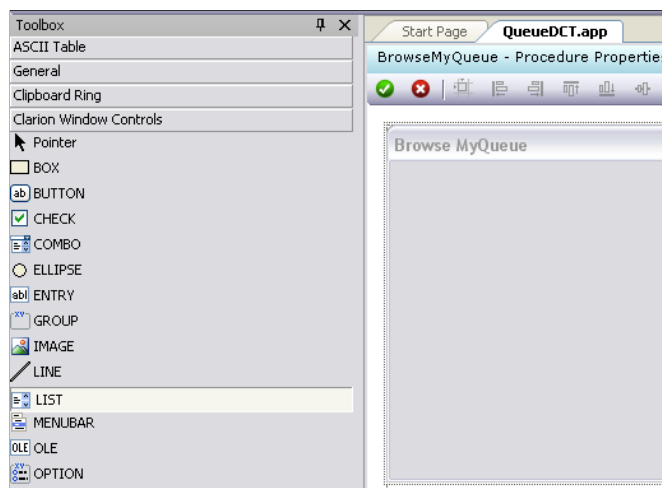


Figure 9. Populating a list "without control template"

Place and size the list box on the window. Then, move to the Properties pad to change the list's "From" property.

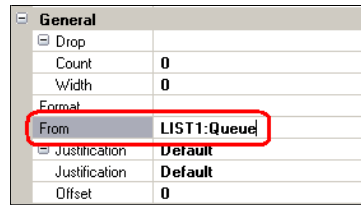


Figure 10. Default "From" property, immediately after populating the list

Click on "From" and select the correct queue:

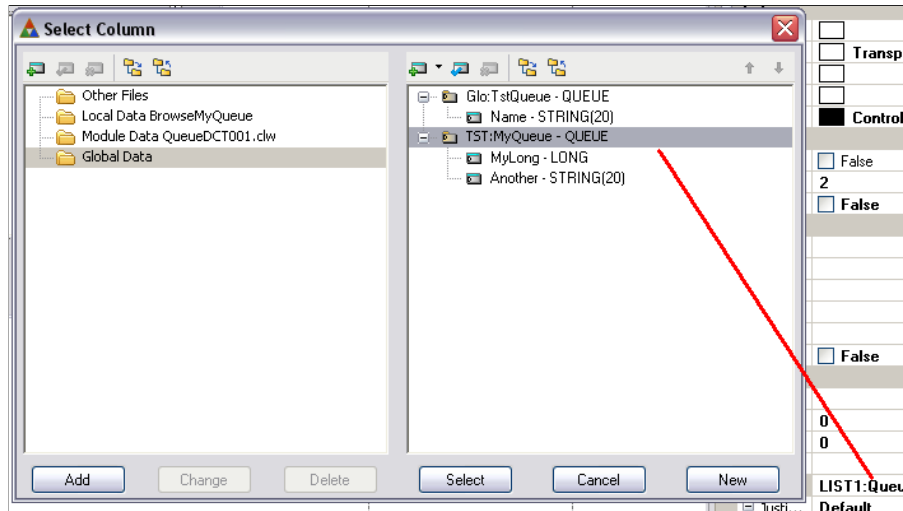


Figure 11. Selecting the desired queue to populate the list

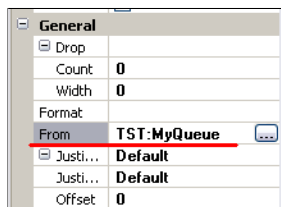


Figure 12. List Properties after selecting source queue

Note: It is still possible to type in the name of a queue, in the case of hand coded queue declarations. Just as in most previous Clarion versions, this works, especially if you just want to display the all the queue elements and don't care about the order of them.

With the list "From" my queue, I can populate the list box with the fields I want:

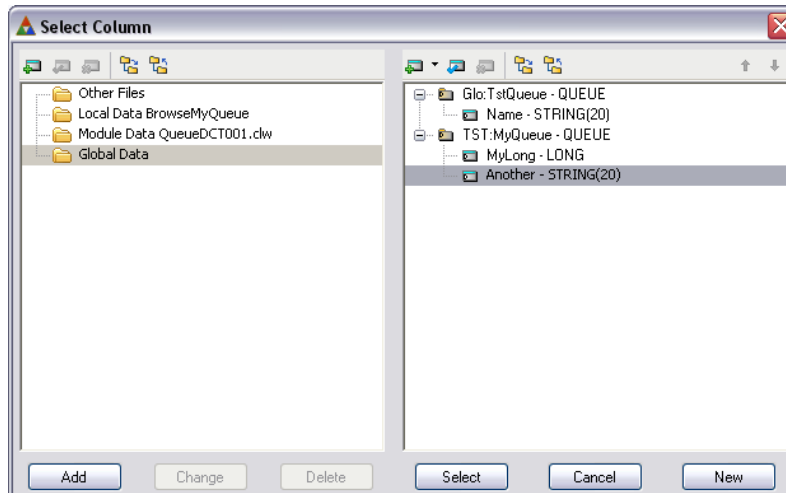


Figure 13. Populating list from queue

The Demo APP

In Main (Frame), I prime each queue so that it will have some values to display:

```
Free(TST: MyQueue)
MyQ: MyLong = 27
MyQ: Another = 'My Record'
Add(TST: MyQueue)
```

```
Free(Gl o: TstQueue)
gTsq: Name = 'This is my Name Field'
Add(GLO: TstQueue)
```

Standard stuff, this.

On the demo app's main menu, I have provided links for the browse to each of the queues. Click Browse TST Queue. You will see that the list box does fill with the value I primed when the app opened:

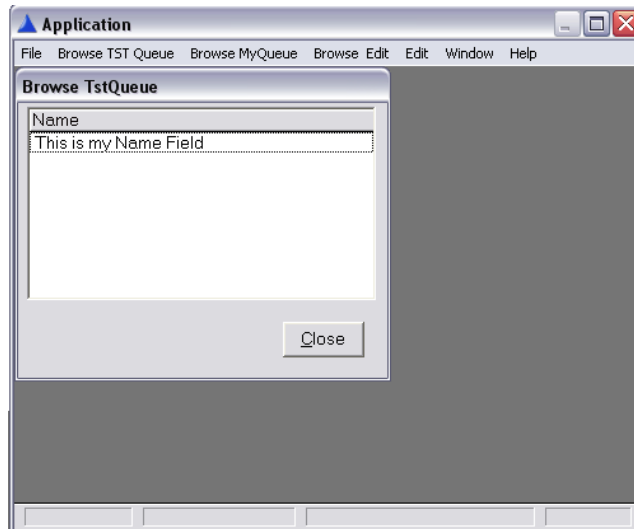


Figure 14. Browse of TSTqueue

Note that, unlike a normal Clarion browse, the first line is not highlighted when opening the procedure.

Adding:

```
?LIST1{PROP: Sel Start} = 1
```

at the end of `INIT` solves this (select `Browse MyQueue` from the main menu):

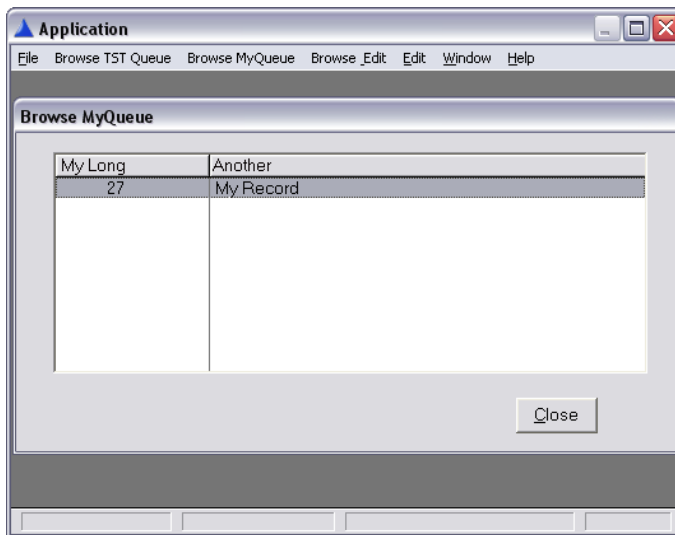


Figure 15. First line selected

Summary

So, there you have it. Dictionary queues are “do-able” and not at all hard.

Yes, the queue and its elements may have different prefixes, unlike hand coded queues. But populating off the data pad, especially now that “click to populate” has been implemented,

is straightforward enough.

Next time: making the queue browse fully editable.

[Download the source](#)

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since version 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.



Article comments

[↑ BACK TO TOP](#)

Copyright © 1999-2010 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.

Welcome To The New ClarionMag Site!

By Dave Harms

Posted June 4 2010

In mid-May of this year we went live with our new ClarionMag site. We've worked hard to retain a familiar user interface, so you may not immediately realize the extent of the changes. But this really is a brand new ClarionMag with many new capabilities and benefits, not all of which may be readily apparent, and not all of which are yet even public.

Some of Clarion Magazine's new benefits include:

- Improved site navigation (you can now page through lists of articles and news items, and your current page is remembered)
- Faster, cleaner page rendering
- Improved, standardized authentication (you can stay logged in if you wish)
- Article ratings, so you can see what other developers recommend
- [Personal article lists](#), including:
 - Unread articles
 - Read articles
 - Favorite articles
 - "Read later" articles
- A "from the archives" random article selection

We'll be posting more detailed explanations of these features in the coming days and weeks.

But this is just the beginning (and yes, we'll have more specific information about these benefits in the weeks ahead). We're also hard at work on something we call The Clarion Roadmap (still in beta). The Clarion Roadmap will make it easier for you to find not just Clarion Magazine content, but Clarion content all over the world wide web. We're in the process of making Clarion Magazine the home of *everything* Clarion.

And if that's not enough, we have other features in the works which we're not prepared to announce just yet. But if you get the idea that ClarionMag is now in a constant state of evolution, then you have the right idea.

Please note that Clarion Magazine's [subscription rates will be increasing on July 18, 2010](#).

About the technology

This is the third major release of Clarion Magazine. The first version was a mostly static site, but we pretty quickly moved past that to a dynamic site running on MySQL and written in Java. That code base served us well for about a decade, but it was getting a bit dated. About three years ago we began evaluating new platforms, focusing on Windows MVC (Model-View-Controller) web app frameworks. Windows, because we hoped to use Clarion#; MVC because I really like the architecture and had used it in the Java version.

Initially we used Clarion# with the Monorail framework and some custom templates, and that worked well, but Microsoft was ramping up the ASP.NET MVC product. MS has really produced a fantastic MVC toolset (which owes not a little to Monorail and to Ruby on Rails), and their offering eventually became too compelling. If you're doing MVC work in Windows, ASP.NET MVC is the overwhelming favorite.

Early on Clarion# had some issues with ASP.NET MVC so we moved to C#, but we're still using C7 to generate a bunch of the C# code used in the ClarionMag site. We also switched databases from MySQL to PostgreSQL.

I remember Mark Riffey once commenting on how a rewrite of an aging, mission-critical application can breathe new life into a company. That's how we feel about the new ClarionMag content delivery system. And this rewrite goes well beyond this publication; we're also using it for our soon-to-be-launched .NET site, DevRoadmaps.com.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of [Developing Clarion for Windows Applications](#), published by SAMS (1995), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).

Article comments

by [Russell Eggen on June 10 2010 \(comment link\)](#)

Testing comments.

by [Dave Harms on June 10 2010 \(comment link\)](#)

And, in fact, comments work.

A few points to keep in mind:

- You need a blank line between paragraphs of text
- You can tag text as source code
- You need to add a space after any < character or it'll be taken as a tag. Only tags corresponding to the toolbar formatting options are allowed, all others are stripped out.

- Be sure to look at the preview to be sure your message is displaying as expected.

 [BACK TO TOP](#)

Copyright © 1999-2010 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.

Editable Dictionary Queues

By Steven Parker

Posted June 15 2010

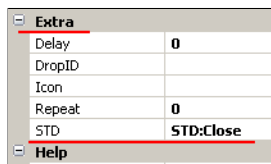
In several previous articles, I have alluded to being able to treat a queue like a file. Using the queue specific semantics of the Clarion language, I have implied that a browse-form can be built for a queue.

In my [last article](#), based on an example from Tony Robinson, I showed how to declare queues in the dictionary. I also showed how to create a fully configurable browse using the standard List Box Formatter, with all its options. It is now time to complete the picture and show how to make a fully functioning browse and form to maintain a queue.

The browse

I constructed my queue browse based on the Window Template. When I opened the Window Formatter, I populated a button to close the window.

Previous versions of Clarion allowed me to select standard window actions – e.g., STD:Close – to associate with the button. That list of standard actions has moved to the window's Properties Pad:



Extra	
Delay	0
DropID	
Icon	
Repeat	0
STD	STD:Close
Help	

Figure 1. Finding "Standard Close"

The next thing I need are buttons to add, update or delete queue records. So, I put three buttons on the window:

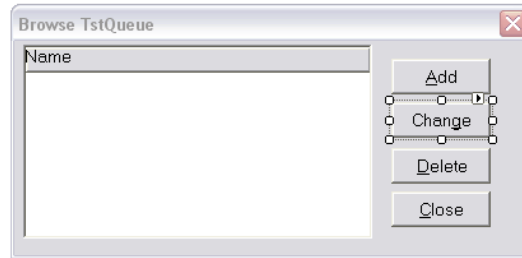


Figure 2. "Standard" buttons

By default, the Windows Controls Pad will generate button Use variables with helpful LABELS like ?BUTTON1 and ?BUTTON2. This would make finding their embeds something less than totally easy. In order to generate embeds with labels that are easy to find, I change the Use variable of each button to match its purpose:

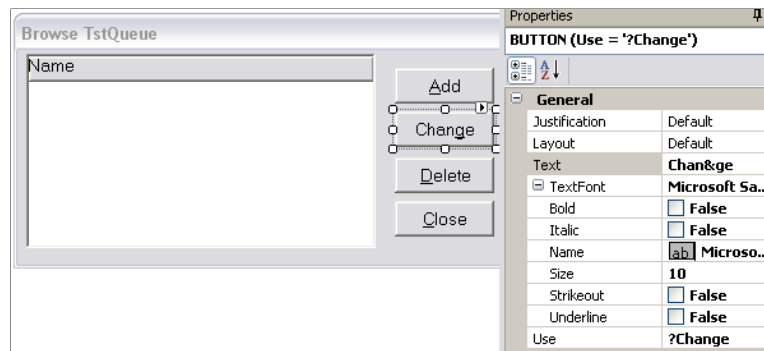


Figure 3. Updating Use variable

I now have my browse window. The browse is working. The buttons are not. And that means that the window is not working, not doing what it's supposed to.

Just what do I need the window to do?

It's all in the spec

First, I need the window to respond to key presses for the Insert, Delete and Enter keys. Note that it is the window not the buttons that needs to respond to the keyboard.

This means I need to ALERT these keys on the window. The dialog to alert keys is on the window's Properties Pad:

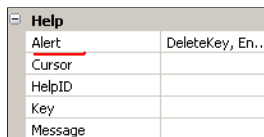


Figure 4. Alert Key dialog for "window"

Right clicking on the window to find this option, as I did for the browse box, will not locate it.

Second, I need to tell the window what to do for each key press. In the window's `AlertKey` embed:

```
Case KEYCODE()  
Of InsertKey  
  Post(Event: Accepted, ?Insert)  
Of DeleteKey  
  Post(Event: Accepted, ?Delete)  
Of EnterKey  
  Post(Event: Accepted, ?Change)  
End
```

correctly emulates the behavior of a template generated browse. (If you're thinking "Why not use the `BrowseUpdateButtons Control Template?`", it's not available.)

Also, as noted in the last article:

```
?LIST1{PROP: Sel Start} = 1
```

late in `ThisWindow`. `INIT` ensures that, when the browse opens, the first line is selected (otherwise, it looks funny).

The buttons

Now I need to make each button do what a template generated update would do. I need to adapt whatever that is for queue semantics.

Insert: An insert expects that the update form receives an empty buffer. For a file, that means clearing the `RECORD` structure. The same syntax is documented and appropriate for a queue. While there is no `RECORD` structure for a queue, the `CLEAR` statement does operate directly on the queue buffer.

Next, set `Global Request` and call the update procedure. In the update form, I will need to know what action the user requested and `Global Request` contains that information. Why? I will need to know whether to `Add(GLO: TstQueue)` or `Put(GLO: TstQueue)`.

On returning from the update form, I select the appropriate queue record and give the list box focus.

This is the final code for an Insert button:

```
Clear(Glo: TstQueue)  
Global Request = InsertRecord  
UpdateTstQueue  
?LIST1{PROP: Sel Start} = POINTER(Glo: TstQueue)  
Select(?List1)
```

The only real difference between this code and the code I would use to manually code an update to a file is

```
?LIST1{PROP: Sel Start} = POINTER(Gl o: TstQueue)
```

Because queues support pointers, reliably, this line ensures that when I return to the browse, the newly added record is highlighted. If the user cancelled out of the form, the last record should be highlighted.

I *could* save the current queue pointer and restore it on a cancelled add. But, that's a bit like work. Code for this might look like:

```
Ptr = Pointer(Gl o: TstQueue)
Clear(Gl o: TstQueue)
Global Request = InsertRecord
UpdateTstQueue
If Global Response = RequestCancelled
  Pointer(Gl o: TstQueue) = Ptr
End
?LIST1{PROP: Sel Start} = POINTER(Gl o: TstQueue)
Select(?Li st1)
```

You may test this for yourself in the sample app which is downloadable at the end of this article. Click “Browse & Edit” on the main menu. (The browse procedures from the previous article are accessible from “Browse (Only).”)

If you run the demo app, you will find that Inserts work, as expected.

Change: To update an existing record, the form expects the target record to be in the buffer. This means that I need to fetch the highlighted queue record.

Global Request needs to be set and the update form called:

```
Get(Gl o: TstQueue, Choi ce(?LI ST1))
Global Request = ChangeRecord
UpdateTstQueue
?LIST1{PROP: Sel Start} = POINTER(Gl o: TstQueue)
Select(?Li st1)
```

This code, too, works as expected.

Delete: Deleting a queue record is also pretty much the same as deleting a file record (but without having to worry about referential integrity).

I need to retrieve the record to be deleted. Then I can simply delete it.

This code does the job:

```
Get(Gl o: TstQueue, Choi ce(?LI ST1))
Del ete(Gl o: TstQueue)
?LI ST1{PROP: Sel Start} = POI NTER(Gl o: TstQueue) - 1
Sel ect(?LI ST1)
```

I could, should I wish, also do an “Are you sure?” check fairly easily:

```
I f Message(' Are you sure?' , ' Confi rm Del ete' , |
  I CON: Questi on, Buton: Yes+Button: No) = Button: Yes
  Get(Gl o: TstQueue, Choi ce(?LI ST1))
  Del ete(Gl o: TstQueue)
  ?LI ST1{PROP: Sel Start} = POI NTER(Gl o: TstQueue) - 1
End
```

In sum: I understand the data requirements for adding, updating and deleting records. That makes setting up I/O for a queue very straightforward. I only need to understand the queue-specific Clarion statements for manipulating queues.

In fact, understanding Clarion’s queue semantics, I could also enable header sorting. Using code from previous articles, I can test for MouseLeft on the header. I can get the column on which the user clicked. Then, I can Sort(GLO: Tst: Queue, ...) and refresh the list.

The form

The “form” is also a simple window. Buttons are added from the Controls pad; entry fields and prompt are selected from the data pad.

Much less is required of the form. In fact, as I see it, the main question is “Did the user complete the action or cancel?”

If the user completed the action, was that action an insert or an update? If the user cancelled, I need to restore the state of the queue.

Cancel: If I need to restore the state of the queue, I first need to know that initial state. I do this by declaring local variables matching the queue’s elements. Early in I N I T, I save current queue record:

```
SAV: Name = gTsq: Name
```

This is the only thing that is a bit harder than it would be with standard files. With standard files, Clarion requires (therefore provides) a group, the RECORD structure, which I can copy:

```
SAV: fi l : RECORD Li ke(fi l : RECORD)
```

and save in one go:

```
SAV: fi l : RECORD = fi l : RECORD
```

Queues do not have a similar structure. So, local copies of the elements must be declared and saved one by one. (A GROUP can be declared inside a queue's definition; so, perhaps it is possible to use such a GROUP as a RECORD substitute.)

In the Cancel button, I re-assign the initial values:

```
gTsq: Name = SAV: Name  
Post(Event: CloseWindow)
```

(Inserts are handled correctly, using this technique, because all the queue elements were CLEARed before calling the form.)

Completed: Because the window clears Global Request, I test the local copy, SELF.Request in the Ok button:

```
Case SELF.Request  
Of InsertRecord  
  Add(Glo:TstQueue)  
Of ChangeRecord  
  Put(Glo:TstQueue)  
End  
Post(Event: CloseWindow)
```

and return to the browse.

If, instead of simply adding the queue record, I want to add records in key order, I can do so:

```
Add(Glo:TstQueue, +Tst: Name)
```

for example.

Open the sample app, add, change, delete records. Everything works and there is no way an end user could know that s/he is working with a queue and not with a file.

Summary

Queues are a uniquely Clarion construct. They also therefore have a special set of access commands.

But, once you understand how queues work, it is entirely possible – and not at all difficult – to create procedures based on queues that are indistinguishable from file-based procedures.

[Download the source](#)

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since version 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.



Article comments

[↑ BACK TO TOP](#)

Copyright © 1999-2010 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.

Everybody Needs Some Closure

By Dennis Evans

Posted June 15 2010

Recently there have been a several discussions about the [functional programming](#) extensions in .NET 3.0. As part of these discussions some people have asked specifically about the functional programming concept of a [closure](#).

So what is a closure?

Simply put, a closure is a function that binds one or more variables to the scope and life of the function. These variables cannot be accessed from outside the scope of the function, they cannot be changed and they persist for the life of the function. That is about as simple and straight forward of a definition of a closure I have ever seen.

In this article I will show you some code written in Clarion# that creates a closure, and I'll examine the IL code generated by the C# compiler to see how a closure actually works within the .NET run time.

I should also tell you what is not going to be covered.

A complete and detailed discussion of closures would require some extensive background on topics like free and bound variables, lexical scoping and a couple of other topics. These topics are covered in various Computer Science text books, web sites and trade periodicals. I am not going to go into these topics, as this is a *quick* look at closures.

I also assume you have had some exposure to .NET, you understand that the specific language used is not important, and you have some familiarity with delegates. If you are not familiar with delegates then you may want to do [some reading on MSDN](#). By the end of this essay you will have a solid understanding of what a closure is and how to create and use one in your programming tasks.

Uses

What are closures good for, and how can you use them in your programs? The obvious answer to that is protecting shared data. When a program has some global data that needs to be shared among different threads a closure is one way to make sure the values are not changed at the wrong time, in the wrong thread etc. Once you write the closure the values

are protected and can be used anywhere in your programs, safely and efficiently. Among other things, closures are an important tool when writing code for multiple CPU cores.

I called closures *functions*, but specifically they are *first class functions*. A first class function can be used as an argument to another function, returned as the value of a function, and used by the code inside a function. Pretty much anything you can do with a simple variable type you can also do with a first class function. Using a function as the return value from a function is basic to all functional languages and one of the more powerful features of functional programming.

One limitation to the functional extensions in C# (and Clarion# is a layer on top of C#) is the closure will not be replaced at compile time. In most functional languages when the compiler sees a closure or an immutable function, the function call will often be removed and replaced with the value. When you write code like `sel f. Global Ids(10)` the compiler will insert the result of the function. Which means at runtime there is no overhead of calling the function. Unfortunately C# does not accomplish this task. Hopefully future versions will correct this issue (F#, Microsoft's .NET functional language, has the same limitation).

Closures and lambdas

In the introduction I said that a closure is a function that binds one or more variables to the scope and life of the function. Another way to say this is that the function is closed over the variables. What is important is that you understand that once a closure is created the variables contained within the closure are immutable, they cannot be changed and the function will always return the same value. The only way to have the function return different values is to destroy the function and recreate it with new values.

In .NET closures are created with [lambda expressions](#).

A .NET lambda expression is an anonymous function that can contain statements and be used to create delegates. Here's a lambda which you can find in the attached example code:

```
f = [x] -> (y + z + x)
```

While that may look a little odd it is really not all that complex.

f is just a variable that is assigned the delegate created by the lambda expression. The lambda expression is composed of two parts, [x] and (y + z + x). [x] is a parameter to the delegate or lambda, y and z are just variables used in the calculation. The types of the variables can be inferred from their use; (y + z + x) indicates that this lambda is going to add three numbers together. When called, the delegate or function f will return the sum of the three variables. Where the values for y and z come from will be clear in a moment.

I am using integers in this example, but the type of the parameters does not have anything to do with creating a closure. The delegate could have been created to concatenate three strings together:

```
f = [x] -> (y & z & x)
```

There is a lot more to lambda expressions, but for the purpose of this essay that should be enough.

The example

If you have not already downloaded the Clarion# source linked at the end of this article, do so now. Compile and run the solution. When the solution runs you will see a screen with three entry controls and two buttons. Enter a number in each of the controls and press the Set Function button. Now press the Show Sum button and the sum of the three numbers entered is displayed. Now change one of the two values labeled Primary Id or Secondary Id and press the Show Sum button. The result displayed does not change to reflect the new values entered. Now change the entry control labeled Add Value and press Show Sum. This time the result displayed will change. The Add Value field is used as the parameter to the delegate, the [x] in the lambda.

Open the MainForm.cln file and locate the event handler for the Set Values button; the method name is btnMakeOneClick. The first two lines just assign the values from two of the text boxes on the screen to the properties of an instance of a class called GlobalClass:

```
self.GlobalClass.PrimaryId = self.tbPrimary.Text  
self.GlobalClass.SecondaryId = self.tbSecondary.Text
```

If this was C# you would be required to convert the string value from the text box to an integer and the code would look like this:

```
GlobalClass.SecondaryId = Convert.ToInt32(self.tbSecondary.Text)
```

The conversion still happens but Clarion# does the work for you.

The next line assigns the the result of a function call to the class member GlobalIds, which is a delegate.

```
self.GlobalIds = |  
    self.GlobalClass.makeClosure(self.GlobalClass.Pid, self.GlobalClass.Sid)
```

The end result here is that GlobalIds can now be used like a function that accepts one input and returns a result. Again, if you are not familiar with delegates in .NET you will need to go to MSDN and do some reading.

If you scroll down a little you will see a line of code that uses GlobalIds:

```
result = self.GlobalIds(addValue)
```

Using a delegate is just like using any other function or method.

At this point you may have a couple of questions. One, the values of the global class properties are changed in the second event handler but the change does not appear when the delegate is called. Two, the class with the method that assigned the Global Ids went out of scope, but the program did not throw an exception. When I get to the IL code both of those questions will be answered but there is a little more Clarion# code that I need to show you first.

Open the source code file ClosureCl.cln. The class contains only one method labeled makeClosure that accepts two integer parameters.

```
ClosureCl.makeClosure procedure(Int32 pld, int32 sl d)

f Func<Int32, Int32>

code
f = [x] -> pl d + sl d + x
return f
```

Not a lot going on in this method but this is where the closure is actually created so I want to spend a little time here and explain what happens. The data section contains the definition of a delegate that will be used as the return value from the method. Remember that this specific use of the Func type defines a delegate which receives a single Int32 value and returns an Int32 value. X is the parameter to the delegate and the local variables pl d and sl d are used as values in the calculation. The y and the z values from the lambda used to create the delegate. When the compiler finds code like this it will create an inner class, sometimes called nested class, and add the two local variables from this method to the inner class as fields. While all that is probably as clear as mud, if you examine some IL code what is happening will become clear.

Go to the Tools menu option and select the IL Dasm menu item. When the IL Dasm displays there will be a node in the tree labeled, closeDemo.ClosureCl. Expand that node and six elements (child nodes) will appear. The first one I want to look at is the one labeled __CLA_LambdaClass_CLA_0, this is the inner class the compiler created. The first thing you should notice is the two fields in the class, pl d and sl d. The compiler took the labels of the local variables in the makeClosure method and used those labels for the fields in the inner class. Below that is a class method, indicated with a purple square and labeled __CLA_Unique_CLA_ID_0. Double click on the node to display the actual IL code. The code I am interested in is listed below:

```
IL_0000: ldarg.0

IL_0001: ldfl d int32 closeDemo.ClosureCl /__CLA_LambdaClass_CLA_0: : pl d

IL_0006: ldarg.0
```

```

IL_0007:  ldfld int32 closeDemo.ClosureCI /__CLA_LambdaCI ass_CLA_0: : sld
IL_000c:  add
IL_000d:  ldarg. 1
IL_000e:  add
IL_000f:  ret

```

The first line of the method, `ldarg. 0`, which means "load argument 0", loads the class reference onto the stack. The next line `ldfld`, which means "load field", loads the class field with the label `pld` onto the stack. The next two lines repeat the process to load the class field `sld` onto the stack. Those two values are then added together. The code now loads the second argument to the method onto the stack and adds that value to the sum of the two class fields. The function will return the result or the sum of the three values.

So far that was pretty simple, the main idea you need from this code is that the class fields, `pld` and `sld` are in use by this delegate. Remember the `y` and `z` values from the lambda? Well this is how those values are provided to the delegate when it is executed. But where do the `y` and `z` values come from in the first place?.

Look down the tree and you will see that `makeClosure` method. Double click on the node to display the IL code. This IL is a little more involved but not too bad and I am only going to look at a few lines. If you are feeling lost, go back to the `Clarion#` code and look at the `closureCI.makeClosure` code.

The first line of the IL creates a new instance of the inner class. The class instance is loaded and then the `ldarg. 1` command is used to load the second parameter, the `pld` parameter, onto the stack. Next the `stfld`, which means store field, line is called. And that primes the `pld` field from the inner class with the value from the parameter. The process is repeated for the `sld` parameter. The remaining lines load a function pointer onto the stack and then call the constructor for the delegate. The delegate is created and then the reference to the delegate is returned to the caller. You should now understand why the values of a closure are immutable and why the function will return the same value. They cannot be changed because there is no way that they can be accessed; those values are completely protected even across threads and you do not need to worry about using synchronization objects.

A couple more points and I will bring this to a close.

The compiler created the inner class and that is what is used by the closure, but not all inner classes are closures. Earlier I said a closure will be created using the local variables of a function and the type does not matter but they must be local. The next block of code looks a lot like the code from the example solution but this code does not create a closure,

```

makeClosure procedure(global CI ass gl )
code
f = [x] -> (ql.PrimaryId + ql.SecondaryId + x)

```

```
return f
```

When this code is compiled the compiler will go through almost the exact process that it went through when it created the closure from the example solution. There is one major difference: the parameter is a class reference and it is not local to this method. The compiler creates the inner class but in this case uses `gl` as the field label and assigns a reference of the class to the field of the inner class. Since the inner class now has a reference to an object there is no way to protect that object from changes. If you need to create a closure with a class then you must use a local instance, like this:

```
makeClosure procedure(globalClass gl)
  g = new globalClass()
  g.PrimaryId = gl.PrimaryId
  g.SecondaryId = gl.SecondaryId
  f = [x] -> (g.PrimaryId + g.SecondaryId + x)
  return f
```

In this example the inner class will still be created with a reference to the class `g` but when the method exits class `g` goes out of scope and you no longer have access. If you are interested you can copy and paste these two examples in the solution, compile it and then look at the IL to see the classes in use.

Garbage Collection

One issue that you need to keep in mind when you are creating closures is that as long as you have a reference to the inner class, typically through the delegate, the outer class will not be picked up by the garbage collector. Normally this will not be worth worrying about as the closure will be created in small working classes, however there will be occasions when a large outer class may be used. If the large outer class uses several other resources then all that memory will not be cleaned up until the last reference to the inner class is discarded. Do not worry about it, but do not forget about it either.

In closing...

Functional programming is not a new concept; it has been around since 1957 when LISP was created. Functional languages have been used to solve a wide range of problems. Some of the more common functional languages include Erlang, F#, LISP, Haskell, ML and Schema.

One of the primary reasons functional languages aren't as widely accepted as imperative languages is because they tend to place a large burden on the hardware. With today's

hardware that burden has vanished. If you have never been exposed to functional programming you should start looking into the subject, and the sooner the better. The concept is simply too important to ignore. Over the next few years the use of functional programming and the intermixing of functional and imperative languages will continue to grow and expand. If you plan on writing code for more than the next five to six years part of the code you write will probably be done using some form of functional programming.

I hope this essay helped you to understand what a closure is and how they are created. In addition, I hope this discussion of closures has aroused your interest in functional programming. You will be using functional programming tools, languages and concepts if you stay in this business much longer; I don't think you will be given a vote.

Using functional languages takes some practice and some time and it is fundamentally different than using imperative languages. Thinking about programming and what tasks need to be accomplished, and not thinking about the implementation details of how those tasks are accomplished, takes a fairly major mind set change. Some experts have made the claim that functional programming will solve the parallel processing code problem. Maybe it will and maybe it won't, but if you look at languages like Erlang they have already made some significant steps in that direction. Personally, I think Clarion programmers will in the near future jump all over functional programming, write less code, and get more work done. Does that ring a bell?

[Download the source](#)

Dennis E. Evans is retired from the U.S. Army. During his time in the military he spent twelve years in the Armored field and eight years in information management. He currently works as an independent contractor and resides in Marion, Illinois with his wife Beverly and their two children Christopher and Jessica. His hobbies include historical simulations, reading and studying different programming languages.

Article comments

 [BACK TO TOP](#)

Copyright © 1999-2010 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.



[Home](#) [Subscribe](#) [E-Books](#) [News](#) [Blog](#) [Store](#) [My ClarionMag](#) [My Lists](#) [Contact](#)

ClarionMag's Subscription Rate Increase Coming July 18

Posted June 16 2010

As of July 18, 2010 our subscription rates will be going up as follows:

One year rates:

- Current subscription renewal: \$150
- New subscription: \$190

Two year rates

- Current subscription renewal: \$230 (\$115/year)
- New subscription: \$280 (\$140 per year)

Three year rates

- Current subscription renewal: \$320 (\$107/year)*
- New subscription: \$360 (\$120 per year)

If you're not sure when your subscription expires, just log on to the [My ClarionMag page](#).

For more information on subscriptions, [click here](#).

To subscribe or renew, go to the [ClarionMag store](#).

* \$106.67

Article comments

by P.G.M. Bouma on June 11 2010 ([comment link](#))

"And in this economy, every dollar counts...." you say. I do not hope that you are going to do this a lot of times.... There are many developers who earn a lot of money, but there are also smaller companies.

by Dave Harms on June 11 2010 ([comment link](#))

There's no doubt that Clarion Magazine's subscription rates are on the way up. But

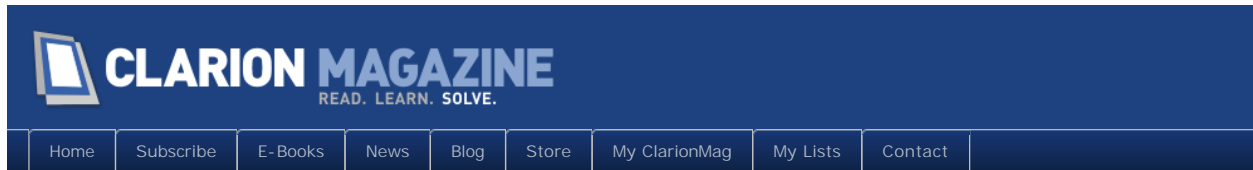
there's also a lot of new value being added to ClarionMag on a regular basis.

My advice to anyone concerned about the price increases is to subscribe/renew now!

If any of you are unsure of when your subscription expires, just log on to the [My ClarionMag page](#).

 [BACK TO TOP](#)

Copyright © 1999-2010 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.



Making a Splash

By Bruce Johnson

Posted June 24 2010

When you have large applications, especially applications with lots of DLLs, it can take quite a bit of time for the program to load, and for the frame screen to appear. Users don't like long periods with no response, so this effect is not ideal. If it takes a really long time for some sign of activity to appear, your users may attempt to start the program again, and end up running multiple instances.

It is misleading, though, to think that this startup time is a single block that cannot be divided. In fact there are two elements that make up this delay, the *load* time, during which Windows is loading your EXE and DLLs into RAM, and the *initialization* time, which is the time from when your app starts running until the application frame window (or other first window) appears.

The goal of this article is partly to offer some suggestions for reducing the load time, but primarily to explain how to display a splash window during the Initialization time.

Rebasing

[Rebasing](#) has been covered in detail in ClarionMag before, most notably by [Carl Barnes](#).

Since Carl wrote that article support for DLL rebasing has been added to the Clarion templates. You can now set the address for each DLL on the Application Global Properties window.

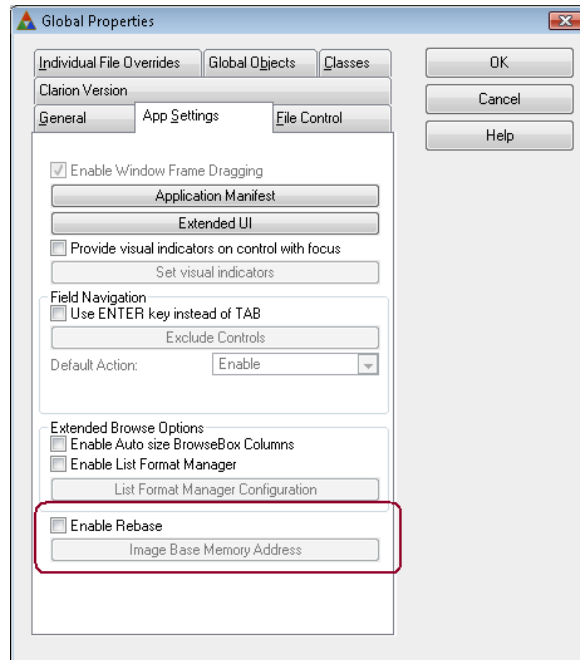


Figure 1. Enabling rebasing

If you are using the [CapeSoft Multi-Proj templates](#) then rebasing is also supported on the Multi-Proj global extension. This is covered in the [Multi-Proj documentation](#). You specify the application number, and Multi-Proj rebases your applications in the order indicated.

Whichever way you do it, rebasing is a good idea. It reduces load time, and can also reduce the resources your app consumes when it is run multiple times on the same PC (for example in a Terminal Services situation) as rebased DLL instances can be shared.

Install the program locally

By installing the program locally on a computer you avoid the overhead of fetching the EXE and any associated DLLs from a network drive. Networks are typically slower than local hard drives, so this can have a big impact on the load, especially if the LAN is weak or overloaded. Installing locally does incur some administrative overhead because now you have multiple copies of your program on the LAN to administer, however tools like [File Manager 3's AutoNet](#) can assist with this.

Initialization time solutions

There may not be much you can do about the time it takes for your application to initialize, but you can change the *perception* of how long it takes. Users want to see something happening; it encourages them to believe that their action (clicking on the shortcut) has been noticed.

Splash Screens

One way to reduce the perceived startup time is to display a splash window to the user, as soon as possible. Then remove this window just before the Frame is displayed.

Clarion has always had the ability to display a splash window, and the Splash window template remains in the shipping templates to this day. However this procedure creates an artificial delay after the program is ready to run, and since this article is about reducing, not extending, the startup time that template is not useful for our purposes.

No, you don't want a splash *procedure*, you need a splash *window*. Something that can be displayed, and which remains visible as the rest of the application initializes normally. If you use a normal Window procedure then the procedure will need to end (and the screen close) before the rest of the initialization can occur.

Create the Global Window Declaration

Because the Window will be opened in one procedure, and closed in another (I'll come to those in a in a bit), the declaration of the window will need to be Global, not Local. This is unusual in the Clarion world, and something you probably have not done before. However it's not very different from creating a local window declaration.

The creation of the global window declaration, as well as the two procedures, should be done in the Data DLL if you are creating a multi-DLL application. If you have a single app file, then the same approach applies, but of course then you just do it in your exe app.

To create the window, go to the global embeds and the GlobalData embed point. In that embed press Ctrl-D (Clarion 7) or Ctrl-F (Clarion 6) to bring up the Window designer. Select the simplest window declaration from the list of defaults (usually this is called Simple Window). Then edit the window until you have the splash window looking the way you want it to look.

The look of the window is up to you, however there are a few things to consider:

- You cannot put any entry type fields on the window. This window will not get any events, and will not have an ACCEPT loop, so it's pretty much a read-only window.
- I recommend removing the caption, and border from the window especially if you plan to use a graphic on the window.
- You can use String controls on the window which can be primed with a specific value (such as the program version number) when the window is opened.
- Set the window to the center position, so that it always appears in the middle of the screen.

Once you have completed your design close the window designer; you should be able to see the code for the window declaration.

In the attached example my declaration of the window, and some other variables, looks

something like this:

```
Global Splash: Open Long
```

```
Global Splash: Text String(40)
```

```
Global Splash: Window WINDOW, AT(, , 243, 72), CENTER, FONT(' Arial ', 10), NOFRAME  
    IMAGE(' splash.bmp '), AT(2, 2), USE(?IMAGE1)  
    STRING(@s40), AT(18, 60, 223), USE(Global Splash: Text), CENTER  
END
```

There are three very important things to note here.

Firstly notice that I've used a string (Global Splash: Text) on the window. I could declare this string anywhere, but for simplicity I've just declared it in code above the window declaration. It's important that whatever way you declare this string it appears before the window in the code or you will get a compile error.

The second thing to notice is the *label* of the window. I've manually changed the label from the generated Window to Global Splash: Window. This is because this declaration is global, and like all global data it needs a unique name. This is very important, if you forget to change this name you'll most likely get a Duplicate Identifier error message when you compile because somewhere in a procedure you'll have a generated WINDOW structure with the label Window.

Lastly notice that there's another global variable, called Global Splash: Open. This will come in handy later on in the two procedures.

Create the procedures

While it would be possible to make the global window exported from the DLL, and simply open it with OPEN(Global Splash: Window) and close it with CLOSE(Global Splash: Window) that approach leads to some possible inefficiencies later on.

So I prefer to create two new procedures in the app file. They are called OpenSplashWindow, and CloseSplashWindow respectively (although you can call them anything you like). These procedures are added to the App file as normal Source procedures. If the App is your Data DLL then tick on the Export Procedure option for both procedures.

Then embed code in both procedures.

The code in the OpenSplashWindow procedure can be as simple as this:

```
If not Global Splash: Open and Thread() = 1  
    Global Splash: Text = ' Version 1.0'  
    Open(Global Splash: Window)  
    Display()
```

To create a source procedure in your app:

1. a) click on the New Procedure button.
2. b) Enter the

```
Global Splash: Open = true  
End
```

The code in the CloseSplashWindow procedure is even simpler:

```
If Global Splash: Open  
  Close(Global Splash: Window)  
  Global Splash: Open = false  
End
```

name of the
new procedure
3. c) Select the
Source
Procedure
option from the
Templates tab

Once you have completed the process and got your splash window working, you can return to these procedures and adjust the code to make it fancier if you like. For example, using WinEvent you may choose to make your splash window semi-transparent (ds_WinTransparent(128)), or you may want to use additional code to prime your strings.

Sidebar: By putting the Open and Close code in procedures, it is possible to tweak the code without triggering a recompile of the whole app. It's also possible to refine the code in a single place to get the effect you wish.

Call OpenSplashWindow

You now have the window, and you have the procedure to open the window. Where is the first possible place you can put the call, so that the window opens as soon as possible? This varies a bit, depending on what sort of system you have.

ABC based, Multi-DLL or single App

If you are building a multi app solution in ABC , then the first line of code that runs is in the Data DLL, not the Exe app. And the earliest place you can embed code in the Data DLL is the embed point called Dictionary Construct. This embed point gets called for every thread that starts, but since you have a thread test in the OpenSplashWindow you don't need to worry about that. In a single app system you use the same embed point, but in the Exe app, not the data DLL.

Legacy Based, Multi-DLL or single App

In Legacy the embed point to open the window is slightly different. In Legacy the first code that runs is in the Exe app file. The correct embed to use here is the Program Setup embed point, Priority 1.

Call CloseSplashWindow

The goal is to call the CloseSplashWindow procedure immediately before any other window in the application becomes visible. Depending on your application there may be multiple windows that may appear first. Fortunately, because there is a simple test inside the

CloseSplashWindow procedure it's completely safe to call it from multiple places - only the first call will do anything.

You're the best person to know which window in your program opens first, so you may need to place the call accordingly, however some of the common places are listed below. Regardless of which procedures use the call, your goal is to place the call to the CloseSplashWindow procedure just before the call to open that first window. In ABC apps that means in the ThisWindow.Init method, after opening the files, but before the window opens. In Legacy apps that means in the Before Opening Window embed point.

So what might be the first screen in your application? Well any of the following might fit the bill;

- The Frame
- Login Screen and CreateFirstUser screen
- Diagnostics screen (like the RuntimeFileManager)
- Other Pre-Frame splash windows.
- First-Run wizard procedures that are called the first time your program runs
- Product Registration window

Feel free to add the call to CloseSplashWindow to all of these, and any others that might appear on startup.

But wait, there's more

The information above is all you need to implement a splash window in your app. There is however one more thing worth mentioning, in case you are feeling in an advanced kinda mood.

During the start of your program, the variables on the splash window can be updated. If you update one or more variables, and then call DISPLAY then the window will be updated.

So if you know why your app is taking a long time to initialize, or you're prepared to put the effort in to find out, then you could update the text on the window during the load process. This will further reassure the user that the program is working.

Indeed if you're really cunning, instead of a string, you could use a progress bar. (Now where have I seen that used before?)

A quick word about threads

As you probably know Clarion 6 and later are preemptively multi-threaded, so using global variables requires some care or strange things may happen. In this case you not only have global variables, you have a global structure as well. So shouldn't you need to use some sort of thread control (for example a Critical Section) around these variables?

The answer is no, because you're only using the structure and the variables on a single thread. In the `OpenSplashWindow` procedure there's a test to make sure the window is only opened on thread number 1.

Final thoughts

The time it takes for your app to initialize is dependent on both the number and size of apps (DLLs) in the system, and the number of files in your dictionary. For small trivial apps (like the attached example) which start very quickly, the splash screen is overkill as it appears and disappears very quickly.

In the example I took the liberty of including an artificial procedure to slow down the initialization process to show the effect you are aiming for.

If your application takes a long time to initialize, then the splash screen can be a useful tool to make the response to the user feel more immediate and limit the number of accidental restarts of your program.

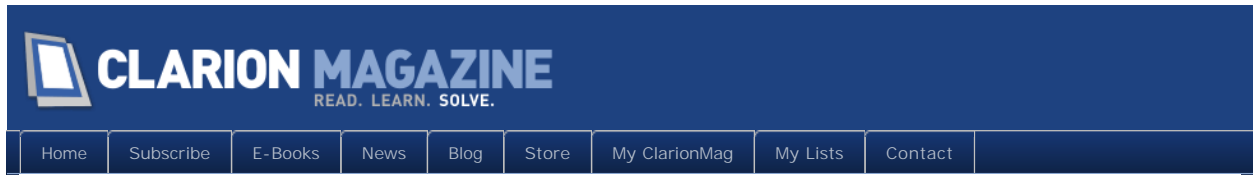
[Download the source](#)

Living in Cape Town, South Africa, Bruce Johnson is a part-owner of CapeSoft and has been programming in Clarion since 1992. He authored the successful "Programming in Clarion's ABC" book and has been involved in some of Clarion's most popular accessories. When not programming he enjoys cooking, and sports - the one as a direct result of the other.

Article comments

 [BACK TO TOP](#)

Copyright © 1999-2010 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.



Queue Record Structures

By Steven Parker

Posted June 29 2010

In [Editable Queues from the Dictionary](#), I used a queue, declared in the dictionary to create a fully editable browse/form combo. However, in the course of that exercise, I found two things that were less than polished, that were sub-optimal:

- in the dictionary, the queue and its elements had different prefixes

and

- when I needed to copy a queue record, I had to save it field by field (the same, if I had to restore the record)

Prefixes

When I created the new Global in the dictionary, the Global Properties dialog requires a prefix.

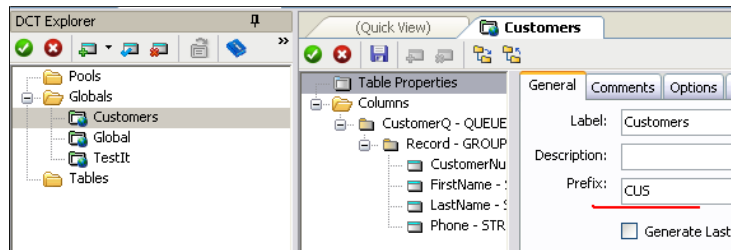


Figure 1. Global, with prefix declaration

Then, when declaring the queue, I gave it a prefix too. I didn't have to, it's force of habit and to ensure that the individual elements have prefixes.

This resulted in code where the queue had a prefix, which, had I declared it in a data embed, it would not have had. Further, the queue's prefix differed from that of its elements.

Dictionary queue declarations resulted in code that looked like this:

```
Free(TST: MyQueue)  
MyQ: MyLong = 27
```

```
MyQ: Another = ' My Record'  
Add(TST: MyQueue)
```

As long as I populate fields from the Data Pad or use Code Completion, this doesn't affect my coding. But, I am more fond of doing my own typing. And even were I not a "type it yourself" kind of developer, the prefix on the queue LABEL and the fact that it is different from the elements' prefix just plain looks funny.

In other words, this isn't wrong. It's just disconcerting.

There is no avoiding the prefix, from the Global declaration, on the queue LABEL. This is, I suppose, part of the price for the convenience of being able to use the dictionary.

So, on a whim, I tried using the same prefix for both the Global and the queue declaration:

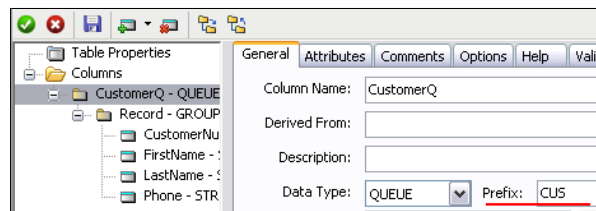


Figure 2. Queue prefix declaration, same prefix

Declaring the same prefix on both the Global and the queue populates from the Data Pad correctly and the code compiles.

Checking Clarion 6.3.9056 demonstrates that this behavior, "re-using" a prefix on a queue previously declared within a Global, does not cause errors (using the same prefix on two items at the Table level does). So, a bit to my surprise, the ability to use the same prefix at the Table and at the Field level, at least in the Globals section, is not new.

Now, my code looks like this:

```
Free(MyQ: MyQueue)  
MyQ: MyLong = 27  
MyQ: Another = ' My Record'  
Add(MyQ: MyQueue)
```

Much nicer, I think. At least it is more consistent.

RECORD structures

FILE structures (for flat files, the SQL term is *tables*) have a mandatory RECORD structure. (It is this structure that provides the record BUFFER.) The RECORD structure is a GROUP-like structure surrounding the file's fields (not keys, indices or memos). This allows declaration of a single variable exactly duplicating the RECORD structure:

```
SAV: Customer Like(CUS: Record)
```

I can use SAV: Customer to save a copy of an entire record in a single statement:

```
SAV: Customer = CUS: Record
```

If I need to restore the record,

```
CUS: Record = SAV: Customer
```

does the job.

Queues, however, do not feature a RECORD structure. This means that when I need to save a queue record, I need to declare a variable for each queue element. Then, I need to assign each queue element to each variable, one at a time. If I need to restore the queue record to its previous state, again, it's one variable at a time.

For larger queue structures, this could involve a lot of embedded code.

Given that this is Clarion, there just has to be another way (maybe better, maybe not but there just has to be an alternative).

In [Editable Queues from the Dictionary](#), I speculated that a GROUP declared inside the queue declaration might be made to serve the same purpose as the RECORD structure does for files and tables.

Prefixes, again

Of course, declaring a GROUP inside a queue and moving the queue's elements into it is easily done. But I immediately discovered that while the queue retained its prefix (from the Global declaration) and the GROUP inherited the prefix of the queue declaration, the individual fields no longer had prefixes.

In other words, the prefix declared on the queue cascaded only to the GROUP and to elements not within the GROUP. Fortunately, I could declare the same prefix I used for the Global and for the queue on the GROUP:

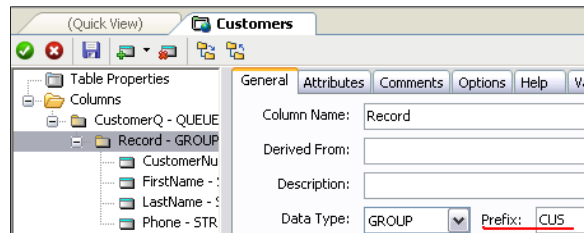


Figure 3. Prefix on the GROUP

and my prefix now attached to the individual elements as expected.

Can a GROUP be a RECORD?

Check the Demo APP. The update form in the demo app does allow me to use my GROUP exactly as I would use the RECORD structure.

I can declare a local variable LI KE the GROUP. I can save the entire queue record to that local variable and I can restore the queue record from the saved copy.

The only problem is that once I set the list box to fill FROM MyQ: MyQueue, when I open the browse, the data does not line up with the column headers:

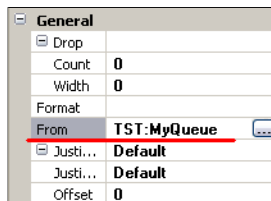


Figure 4. Set List to be FROM global queue

(MyQ: MyQueue is "TST: MyQueue" in this screen shot; it was taken before manipulating the various prefixes). But on the update form, each datum appears in the appropriate field.

In the immortal words of Yul Brynner, "But! Is a puzzlement."

Of course, what is happening is that I have introduced a new column, MyQ: Record, the GROUP, into the equation. The columns don't line up because the List Box formatter does not know the queue's field order. It does know to ignore the RECORD structure and GROUPs for a file or table. But it doesn't seem to do the same for a queue.

To make the list work, I need to turn off AutoFieldNumber and tell the formatter each field's ordinal position:

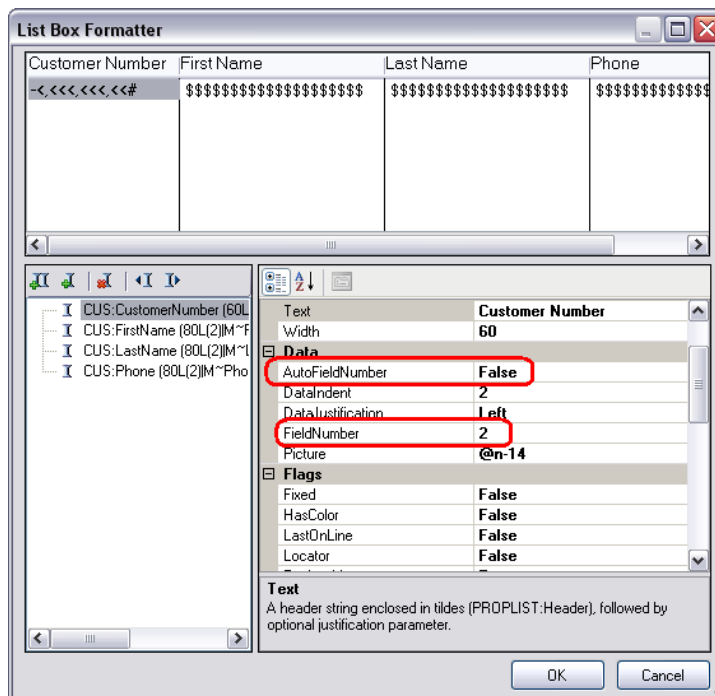


Figure 5. Manually setting field selection

Now everything lines up correctly. Not so hard, really.

Summary

So a GROUP in a queue *can* function like the RECORD structure in a file. *That* is useful to know.

[Download the source](#)

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since version 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.



Article comments

[↑ BACK TO TOP](#)

Copyright © 1999-2010 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.

Clarion 7.2 Makes Steady Progress

By Dave Harms

Posted June 30 2010

On June 3 2010 SoftVelocity released Clarion 7.2 build 7232, followed a week or so later by build 7248. I'll cover both those builds in this article.

Build 7232 was the first C7.2 build, and it came with two important warnings:

1. The APP file format has changed, and
2. All third party DLLs must be recompiled

You cannot convert an APP from the 7.2 format back to the 7.1 format. From the release notes: "When you open your. App file for editing you'll be prompted to confirm that you want to convert it to 7.2 format. If you have a multi-APP Solution, you can initiate a generate all from within the Applications pad, and as each App is opened you'll be prompted to confirm you want to convert to 7.2 format."

And be sure to get updated DLLs as necessary from third party vendors. Failure to do so may result in Bad Things happening to your application(s).

New features

Key new features in C7.2 include:

- A library for integrating the new Report Writer with your apps
- OS theme support for menus (not so much new as backward compatibility with C6)
- In the data pad, a list of procedures used by the template
- Better error handling in the dictionary synchronizer
- A fix for app recovery log performance problems

Glitches in the first 7.2 release

With the initial C7.2 release a number of users immediately encountered a serious performance problem. C7 has an app recovery log feature, and if you use this feature then the IDE writes out some log entries (which, judging by their size, may in fact be an entire

copy of the app) to disk. This feature has been less than performant on some machines, while not causing delays on others, and I don't think anyone's figured out why that is. But in C7.1 you could disable this feature by setting the maximum app recovery log file size to zero.

Happily, eight days later on June 11, SoftVelocity released build 7248 which remedies that problem, and apparently goes well beyond the previous fix.

First, a little background on the app recovery log, a feature introduced in Clarion 7.1. The idea is pretty simple: write out a recovery log whenever any changes are made to the APP file, so that the latest version can be recovered in the event of a crash. Even C6 had an auto-saved backup version, but that only helps you if you've saved your work just before the crash.

The initial implementation of the recovery log left something to be desired. Although the idea of a log would seem to be to write out only the changed information, judging by the jumps in file size the entire app was being written out every time. That's not necessarily a problem, but the data was also being flushed out to the disk in fairly small chunks. On some machines (including, apparently, all of SoftVelocity's) that never resulted in a delay. On my computer, C7.1 didn't so much write the recovery log to disk as it machine-gunned the data onto the platter. At least that's how it sounded, and the hail of data reminded me exactly of some code I once wrote to log debugging information to a file. Not wanting to lose any data, I flushed each write, and as with C7.1 the execution speed of the application running the debug code slowed to a crawl. The solution, in that case, was to flush the data less often; the deafening roar of the hard drive fell into a sweet silence, punctuated by an occasional head seek. And my app's performance returned to normal.

Flushing was suggested by a number of developers as the cause of the problem with C7.1. I don't know if that's specifically what SV addressed, but whatever approach they took the increase in speed in the latest release has provoked expressions of astonishment. Look in the sv.clarion.clarion7 newsgroup for the thread titled "Holy Crap!", started on June 11.

So it looks like this one is taken care of.

Another problem noted in build 7232: locators for the app tree and embed tree dialogs were broken. That issue was corrected for build 7248.

New and used bugs

Robert Paresi has reported a problem with newly added procedure data being lost, and I've seen this too. Ben Dell reported that if you make a change to the procedure settings the added local data will be preserved.

You may need to be careful when using the Esc key to exit the window formatter. As noted by the ever vigilant Mr. Paresi, the following sequence can hang the IDE:

- Open Application
- From App Tree, highlight window procedure and hit WINDOW button
- Hit ESC key to exit Window Designer
- Press OK

Technically it's the Yes button, but in any case I've been able to reproduce this problem on one occasion, but not since. Lee White notes that doesn't happen on XP-Pro

There have been reports of a runtime bug in 7.x which causes menus to disappear from running applications. Again, I've had this happen to me once.

Blog entries

SoftVelocity has posted a couple of 7.2-related blog entries.

The [release announcement](#) includes a link to the [readme](#) as well as some screen shots of the new report writer.

The [Coming up with 7.2](#) post covers the restoration of the Procedures list to the IDE; you can find it on the Data Pad.

There's a brief post on the [ReportWriter and the report engine](#); the new ReportWriter has been out for a while, but this release includes improvements such as support for the older TXR format.

Finally, there's a [post](#) on support for OS menus, including a number of screen shots.

OS menus

One of the really important fixes in 7.2 is the restoration of operating-system styled menus. Prior to C7, Clarion apps used the operating system menus; C7, with much fanfare, introduced themed, owner-drawn menus, to the dismay of many developers. The controversy is more than a year old - see the [Clarion 7 Build 5615 Review](#) for an overview of the issues.

As of C7.2, you again have the ability to let the operating system decide what your menus should look like. But there seem to be a few wrinkles left to iron out, such as the sub-menu indicator turning white when selected instead of staying black.

The new way to do old things

There are a lot of little things in the C7 IDE that are different from the C6 IDE. Sometimes worse, often better, but it's the difference that trips folks up. A case in point is selection of multiple controls. In C6, if you do this with, say, the intent to align those controls, you have to make sure you select the reference control (the one to align to) last.

In C7, the reference control is the first control you select, and that difference can get really annoying if your habit is deeply ingrained. But there's a twist: in C6 the reference control is fixed; in C7, once you've selected multiple controls you can click again to select the reference control, either with the right mouse button or the left mouse button. That's quite useful.

Summary

Clarion 7.2 continues to make progress and win over C6 developers. It's not perfect, but it's good enough for more and more of us. Steady as she goes.

How's 7.2 working for you? Add your comments below.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of Developing Clarion for Windows Applications, published by SAMS (1995), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).

Article comments

by Robert Hutchison on July 1 2010 ([comment link](#))

I wasted a good few days earlier in the year and eventually gave up on Clarion 7 for the moment. I just don't have the time to waste and would prefer to wait to hear that a bullet proof version is there for the using. In my opinion Clarion 1.5 2.0 4.0 5.0 (never used 5.5) and 6.0 were real quality.

"It's not perfect, but it's good enough for more and more of us. Steady as she goes."

Still sounds to me like its not Gold yet never mind 7.2

Just as well I have learned to be patient.

regards

by Dave Harms on July 2 2010 ([comment link](#))

There are developers like yourself who don't want to touch C7 until it's rock solid (leaving aside the point that C6 has its share of bugs too - we've just all learned to live with/work around them). And there are developers who have been using C7 for more than a year and are happy with it.

I think there's much more to be said in favor of C7 than against it, at this point.

 [BACK TO TOP](#)

Copyright © 1999-2010 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.

Queue Browsers, Sort Headers and Locators

By Steven Parker

Posted June 30 2010

In my [last article](#), I showed how to create and use a queue equivalent of a file's RECORD structure. Prior to that article, I showed how, using Clarion's queue semantics, I could construct a standard browse-form pair.

There are two final functionalities I need to implement to make my queue-based list and form completely mirror the browse-forms options provided by the templates:

- Sort Headers

and

- Locators

I am fully convinced that knowledge of Clarion's queue manipulation statements should be sufficient to implement these two features.

Sort Headers

From previous articles, I know that I can detect where a user clicks MouseLeft. In those cases, I can sort my queue with this code:

```
IF KEYCODE() = MouseLeft
  IF ?LIST1{PROPLIST: MouseDownRow} = 0 ! the header
    CASE ?LIST1{PROPLIST: MouseDownField}
      OF 1
        SORT(CUS: CustomerQ, CUS: CustomerNumber)
      OF 2
        SORT(CUS: CustomerQ, CUS: FirstName)
      OF 3
        SORT(CUS: CustomerQ, CUS: LastName)
      OF 4
```

```
Sort (CUS: CustomerQ, CUS: Phone)
End
End
End
```

Two questions:

1. How do I reset the List Box to display the new sort order
2. Where does this code go?

To get a bit ahead of the story, resetting the queue is unnecessary. I had been thinking of using the `ResetQueue` method and, if that didn't work, that old standby, `ThisWindow.Reset(1)`.

Well, it turns out that neither of these is necessary. Once I issue the `Sort` statement, the list refreshes itself. The new sort order is displayed without further intervention from me. That's because the queue data is "bound" to the listbox with the `FROM` statement.

Well, that was easy.

So, where do I embed this code?

Previous experience showed that enabling `Sort Headers` by-passed code for `ALERT (MouseDown)`. This led me to conclude that the place for this code was in `ThisWindow.TakeEvent`.

`ThisWindow.TakeEvent` does work. But my conclusion is not entirely correct for a queue-based browse.

In `ThisWindow.TakeEvent`, using the abbreviated code shown below, I picked up the `MouseDown` click correctly. What I didn't get was the `STOP` on `?Browse: 1 {PROPLIST: MouseDownRow} > 0`.

```
If KEYCODE() = MouseLeft
  Stop('mouse left pressed')
  If ?Browse: 1{PROPLIST: MouseDownRow} > 0
    Stop('List Selected on MouseDownRow')
  End
End
```

Well, examining this snippet more closely now, I realize I had fat-fingered the code. I was incorrectly testing for `MouseDown` on the List Box, not the header. Changing the third line of the code to:

```
If ?Browse: 1{PROPLIST: MouseDownRow} = 0
```

resulted in code that worked.

But, there is an upside to my error. It made me go looking for another place to embed my code.

And then it hit me. The problem I had experience with enabling Sort Headers by-passing code for ALERT(MouseLeft) turned on the List Box having been populated by the templates. I showed that Sort Headers is a template feature and does generate code that short circuits This Window. TakeEvent.

But my current Browse Box is *not* a template browse. It is a simple control.

In this case, I can ALERT(MouseLeft) on the list and use the AlertKey embed for my Sort code. Indeed, this does work. In the demo app that accompanies this article, I did ALERT(MouseLeft) on the list and my Sort code is in the AlertKey embed.

Locators

Getting a specific queue record is easy. Prime the "key" element(s) and perform a GET (queue, key). The docs say this form of Get:

Searches for the first QUEUE entry that matches the value in the key field(s). Multiple key parameters may be used (up to 16), separated by commas. If the QUEUE has not been SORTed on the field(s) used as the key parameter(s), the key indicates an "alternate sort order" which is then cached (making a subsequent SORT on those same fields very efficient).

I created a local variable, FindNbr, to use as a locator on CUS: CustomerNumber. When the FindNbr entry field is completed (this is an entry locator):

```
Get(CUS: CustomerQ, FindNbr)
If ~ErrorCode()
    ?LIST1{PROP: SelStart} = POINTER(CUS: CustomerQ)
    Select(?LIST1)
End
Clear(FindNbr) ! clear "Locator" whether error or not
Display(?FindNbr)
```

If the value is found, I move the highlight bar to the queue record and re-select the list.

I clear the locator field, making it available for another entry.

That's all it takes.

An incremental locator, I suppose, is also do-able. But it would be a lot more work, as would a locator that located on a partial entry. That's less important than the fact that both incremental and partial location appear to be possible.

Summary

There is no question in my mind that I could add tabs to the browse in the demo app. I could

easily embed code in TakeNewSelecti on to display different sort orders, just like a standard browse. Years ago, I showed how to do this, without any pre-existing key, for files using SetOrder. Sort does for queues what SetOrder does for files.

If I can do it for one, I can do it for the other. And, in a very real sense, that's what I've been demonstrating in these articles: if the templates can do *<whatever>* for a file, I can do the same using a queue.

[Download the source](#)

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since version 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.



Article comments

[↑ BACK TO TOP](#)

Copyright © 1999-2010 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.