Clarion Magazine

Clarion News

- » StrategyOnline March Promotion
- » CHT Utility Applications 4.02
- » 1st Logo Design Newsletters
- » SetupBuilder 6.7 February 2008 Update 2
- » DMC Certified For Windows 2008 Server
- » Clarion Handy Tools Blog
- » FullRecord 2.08
- » Clarion.NET Examples Updated
- » Aussie DevCon Date Change
- » Clarion# Invoicing App Progressing
- » ABC Free Templates and Tools Updated
- » Gitano Discontinuing Clarion Utilities
- » UltraTree, HyperBrowze, Up & Up Sale, Up to 50% Off
- » DMC 1.1.0.6
- » vuSendKeys 1.3
- » CHT Utility Apps
- » EasyCOM2INC ver 2.08
- » Clarion Developer Meeting In Pietermaritzburg
- » Replicate Updated
- » FM3 Updated
- » Secwin Updated
- » RightReports Updated
- » Insight Graphing Updated
- » PostgreSQL 8.3
- » SetupBuilder 6.7 February 2008 Update
- » vuSendKeys 1.2
- » Clarion# Browse With Datagrid
- » C7 RTL Update Targets Visuals
- » Clarion# Examples
- » Handy Tools 12A1.01
- » Gitano Releases Sterling Collection
- » FullRecord 2.07 and 1.86
- » vuSendKeys 1.1

[More news]

- » Clarion.NET FAQ
- » Clarion# Language Comparison
- » Clarion# Array Index And Class Instantiation Changes

[More Clarion & .NET]

[More Clarion 101]

Latest Free Content

- » Source Code Library 2008.01.31 Available

[More free articles]

Latest Subscriber Content

Creating A Drag & Drop Batch Compiler

DDE is still the only way to control the Clarion IDE with an external application; Richard Rose shows how to combine DDE and drag and drop to create an easy to use batch compiler.

Posted Friday, February 29, 2008

Understanding Clarion# Strings: Revisited

The latest changes to class instantiation and array indexes in Clarion# have implications for string handling. Dave Harms takes a close look at the ClaString and String data types.

Posted Friday, February 29, 2008

Clarion# Array Index And Class Instantiation Changes

Build 2957 of Clarion.NET introduced some significant changes to the Clarion# language, including the removal of automatic class instantiation and a switch to zero-based arrays. Dave Harms looks at the reasons behind the changes and the impact on Clarion# code.

Posted Wednesday, February 27, 2008

Getting Useful Information Out of SQL, Part 3

In this third installment in his SQL series Steve Parker answers the question "How do I retrieve multiple rows from Prop:SQL?"

Posted Friday, February 22, 2008

Getting Useful Information Out of SQL, Part 2

In this second of three parts, Steve Parker explores the magic of, and the misinformation about, the dummy table technique.

Posted Thursday, February 21, 2008

Getting Useful Information Out of SQL, Part 1

What does SQL offer the discerning Clarion developer, anyway? In this first of three parts, Steve Parker casts a skeptical eye on some of SQL's supposed advantages, but still finds a motivating use for SQL.

Posted Friday, February 15, 2008

Ergonomics For Programmers

All too often programmers ignore ergonomics when setting up an office. They'll spend time and energy researching what is the best computer system, which monitor to choose and which software will do the job, but often don't give much, if any, thought to the physical stresses they'll experience if their desk, chair, monitor, keyboard, or mouse are incorrectly positioned. Robert Johnson explains how to set your office up the right way.

Posted Tuesday, February 12, 2008

Source Code Library 2008.01.31 Available

The Clarion Magazine Source Code Library has been updated to include the January source. Source code subscribers can download the Jan 2008 update from the My ClarionMag page. If you're on Vista please run Lindersoft's Clarion detection patch first.

Posted Tuesday, February 12, 2008

[Last 10 articles] [Last 25 articles] [All content]

Source Code

The ClarionMag Source Code Library

Clarion Magazine is more than just a great place to learn about Clarion development techniques, it's also home to a massive collection of Clarion source code. Clarion subscribers already know this, but now we've made it easier for subscribers and non-subscribers alike to find the code they need.

The Clarion Magazine Source Library is a single point download of all article source code, complete with an article cross-reference.

More info • Subscribe now

Printed Books & E-Books

E-Books

E-books are another great way to get the information you want from Clarion Magazine. Your time is valuable; with our e-books, you spend less time hunting down the information you need. We're constantly collecting the best Clarion Magazine articles by top developers into themed PDFs, so you'll always have a ready reference for your favorite Clarion development topics.

Printed Books

As handy as the Clarion Magazine web site is, sometimes you just want to read articles in print. We've collected some of the best ClarionMag articles into the following print books:

Clarion Sites

Clarion Blogs

- » Clarion Tips & Techniques Volume 4 - ISBN 978-0-9784034-09
- » Clarion Tips & Techniques Volume 3 - ISBN: 0-9689553-9-8
- » Clarion 6 Tips & Techniques Volume 1 - ISBN: 0-9689553-8-X
- » Clarion 5.x Tips and Techniques, Volume 1 - ISBN: 0-9689553-5-5
- » Clarion 5.x Tips and Techniques, Volume 2 - ISBN: 0-9689553-6-3
- » Clarion Databases & SQL - ISBN: 0-9689553-3-9

We also publish Russ Eggen's widely-acclaimed Programming Objects in Clarion, an introduction to OOP and ABC.

From The Publisher

About Clarion Magazine

Clarion Magazine is your premier source for news about, and in-depth articles on Clarion software development. We publish articles by many of the leading developers in the Clarion community, covering subjects from everyday programming tasks to specialized techniques you won't learn anywhere else. Whether you're just getting started with Clarion, or are a seasoned veteran, Clarion Magazine has the information *you* need.

Subscriptions

While we do publish some free content, most Clarion Magazine articles are for subscribers only. Your subscription not only gets you premium content in the form of new articles, it also includes all the back issues. Our search engine lets you do simple or complex searches on both articles and news items. Subscribers can also post questions and comments directly to articles.

Satisfaction Guaranteed

For just pennies per day you can have this wealth of Clarion development information at your fingertips. Your Clarion magazine subscription will more than pay for itself - you have my personal guarantee.

Dave Harms

ISSN

Clarion Magazine's ISSN

Clarion Magazine's International Standard Serial Number (ISSN) is 1718-9942.

About ISSN

The ISSN is the standardized international code which allows the identification of any serial publication, including electronic serials, independently of its country of publication, of its language or alphabet, of its frequency, medium, etc.

Clarion Magazine

Clarion News

Search the news archive

dpQuery 2.10

dpQuery 2.10 is now available. Changes include: New methods (AfterChangeQuery, AfterSaveQuery, OnTabChanged and LoadData); New Embed point (dpQuery SaveRecord method -> On success); New example which shows using of LoadData() method in the DEMO application; Selecting other driver from the list will automaticaly show a Choose Data Source dialog. Free for all customers with current subscription. Demo available.
Posted Wednesday, March 05, 2008

EZRound Tutorial

Charles Edmonds has a new tutorial for EZRound showing examples of using it to create CSS based web site designs.
Posted Wednesday, March 05, 2008

Oak Park Servers Upgraded

Oak Park Solutions has upgraded its Virtual Dedicated Servers to provide more options like increased RAM, etc. and will be running a sale on Virtual Dedicated Servers for as little as $29.99 per month with no long-term commitment for Clarion Developers and their clients. Oak Park also offers new domain names or transfers for as little as $6.95 and web hosting for as little as $4.99 per month.
Posted Wednesday, March 05, 2008

Wingnut Solutions Acquires Gitano Utilities

Dave Hlavac/Wingnut Solutions Inc. has acquired all of Gitano Software's Third Party Utilities. This includes gCal, gCalc, gNotes, gFileFind, gQ, gSec, gReg, and LGP. A support forum is now available, and online purchasing is coming soon. Wingnut has also purchased Berthume Software's cpTracker Pro, so you can expect to see some new features and integration with gReg.
Posted Wednesday, March 05, 2008

RightReports 1.06

New in RightReports 1.06: For ImportTXR, imported user variables now have prompts assigned to them; Improved MySQL support (please see Version History for more details). Requires latest version of xFiles to support import/export.
Posted Wednesday, March 05, 2008

xFiles 1.65

New in xFiles 1.65: Added two new procedures, AddOldField and LoadAllOldFields. This allows you to change a table, view, group or queue field name and still import and XML file using the old field name.
Posted Wednesday, March 05, 2008

FM3 4.30

New in FM3 4.30: New function ds_CreateSchema (for MSSQL driver maintained applications) creates a non-default schema; If TPS create file fails (not open) then replace the old file; Default SQL_Connect procedure prompt in FM3 template to SQL_Connect; Don't hide the Schema name for MSSQL driver applications; New temnplate prompt 'Don't show current file-check window during full upgrade' - if checked will not sure any yellow windows during application startup if there are no files to upgrade.
Posted Wednesday, March 05, 2008

Secwin 4.44

New in Secwin 4.44: Optimize DLL (ds_GetUsers) makes OperatorBrowse faster; Template change forces " into email address template field if non-existent; A switch on the Create_Secwin_Menu template allows quick disabling of AccessControl menu items; If a blank file exists in the Files to Include (on the SecwinSetGlobalAccess window), ignore it; Several TXA changes.
Posted Wednesday, March 05, 2008

Draw 2.61

Changes in Draw 2.61 include: Added support for FreeImage 3.10; Fixed unresolved externals error when compiling in multi-DLL mode.
Posted Wednesday, March 05, 2008

Replicate 2.41

Changes in Replicate 2.41 include: Fix to GPF in critical section while another thread was in a wait. CriticalSection is changed from pointer to a class, removing the need for new and dispose.
Posted Wednesday, March 05, 2008

Office Inside 2.63

Changes in Office Inside 2.63 include: Added ImportSetup() and ImportComplete callback methods to allow the document to be modified before and after the actual import when using the ImportExcel() method; Fixed a problem with the Report To Excel template displaying a prompt to save the current sheet when used in versions of Office prior to 2007; Added code for Excel and Word editable reports to check the version of Office being used do the file saving appropriately
Posted Wednesday, March 05, 2008

New StrategyOnline Web Site

The new StrategyOnline web site features a blog-like dynamic home page with updated content and the option to post comments. The new site also features a shopping cart, but you can still buy via eSellerate and ClarionShop if you wish. Also on the new site: new forums and a case tracker.
Posted Wednesday, March 05, 2008

DCT2SQL Templates Updated

The latest DCT2SQL templates include a number of enhancements to the PostgreSQL code. There are two new PostgreSQL utilities (RESET and VACUUM). Each utility will create a procedure that will open each file, perform the operation, and close each file. VACUUM removes deleted records. RESET will delete all the data from the files and reset the SEQUENCE number back to 1. The UTIL_DictionaryFiles4 utility works under the assumption that all your relationships have been defined in the dictionary. If you have done so, you will be rewarded with a count of parents

and children in the listing. There is also an option to generate skeleton procedures for all tables so you can add sample data; these will generate a calling procedure that will call everything in the right order. If you change the relationships you can re-generate the calling procedure and everything will be in the right order again. Jimmy Rogers contributed his Firebird conversion templates to the collection. There are six new templates with the prefix JKR_. The templates have been added as they are, and might require a tiny bit of work to run under 6.3.
Posted Wednesday, March 05, 2008

Clarion FreeImage Project Update

This release of the Clarion FreeImage Project adds support for version 3.10.0 of the FreeImage library. It adds support for reading and writing JPEG-2000 File Format (*.JP2) and JPEG-2000 codestream (*.J2K, *.J2C) images, ILM piz-based wavelet format OpenEXR files, and fixes a number of issues. Please see freeimage.sourceforge.net for a complete change list. You must download the new FreeImage.DLL from http://freeimage.sourceforge.net for use with this release. This version adds new screen capture methods CaptureScreen(Window), CaptureScreen(hwnd), and CaptureScreen(hwnd, rect). Also fixed are a GPF when saving an image to a BLOB when using Windows Vista, and an error in the Save() method where the FIF was not set to file format selected in the file dialog. This only occured when the image had no file name as is the case when it's created with NewImage().
Posted Wednesday, March 05, 2008

Clarion Desktop 4.08

Clarion Desktop 4.08 is available for download. This is basically a maintenance release, although it does include a link to Gus' new blog.
Posted Wednesday, March 05, 2008

SetupBuilder Italian Module

Thanks to Guennadi Iounok of Motleysoft.com (www.motleysoft.com), an Italian language module is available for SetupBuilder 6.7 Build 2153. If you are interested in the Italian Language Module, please download and install the update (167 KB).
Posted Wednesday, March 05, 2008

StrategyOnline March Promotion

To celebrate the redesign of its web site, StrategyOnline is running a promotion for the month of March. Everyone who purchases anything on the company site between February 26 and March 31, 2008 will receive a three month license for Clarion Desktop. One person will be selected to win a free license to any one StrategyOnline product.
Posted Wednesday, February 27, 2008

CHT Utility Applications 4.02

New versions (V 4.02) of the following utility applications are now available: CHT Handy Zip'n Email; CHT Handy Zip'n FTP; CHT Handy Zip'n HTTP Post; CHT Video Education Player. The CHT tool kit price to new and expired subscribers was lowered from $499.00 US to $299.00 CDN, Nov 1, 2007 as part of the "More CHT Subscribers Wanted" campaign. Feb 29th is the final day of that lower price. March 1, 2008, it goes back to $499 CDN. Renewal prices will remain as always, at $250.00 for a single year. Multiple years, cheaper still and special pricing for multi-developer customers.
Posted Wednesday, February 27, 2008

1st Logo Design Newsletters

1st Logo Design newsletters keep you informed about new products, special offers and other related news. Be notified of special deals, last minute low-price offers and much more. Sign up now and get a free XP icon collection.
Posted Wednesday, February 27, 2008

SetupBuilder 6.7 February 2008 Update 2

Lindersoft has released SetupBuilder 6.7 "February 2008 Update 2". This release is available, free of charge, to all SetupBuilder customers who have an active SetupBuilder maintenance subscription plan. This maintenance release contains fixes that are designed to correct known issues and to improve the overall functionality of SetupBuilder. To get the latest product version, select "Check for Updates" from within the SetupBuilder 6 IDE. To get the latest documentation, select "Check for Documentation Updates" from within the SetupBuilder 6 IDE.
Posted Wednesday, February 27, 2008

DMC Certified For Windows 2008 Server

Thanks to a change in Microsoft's testing procedure, Data Management Center has been officially approved by Microsoft as certified for Windows 2008 Server.
Posted Wednesday, February 27, 2008

Clarion Handy Tools Blog

Clarion Handy Tools has a new blog called "The CHT Blogger". The blog will be used to publish What's New information about The Clarion Handy Tools as a a more public extension of the information already provided in the user forum and on the monthly What's New Page. An RSS feed is available.
Posted Wednesday, February 27, 2008

FullRecord 2.08

FullRecord 2.08 is now available. Changes include: Procedure name now can be recorded in "source" type procedures; ABC template fixed, was not compatible with Clarion 6.1 and before; Procedure name generation moved to first priority on Init code section; Read buffer was incorrectly saved when a change operation was taken place right after an insert - fixed.
Posted Wednesday, February 27, 2008

Clarion.NET Examples Updated

Kevin Erskine has begun coding his existing Clarion 6.3 classes to Clarion#.Net; he plans to release them all as public domain over the coming months.
Posted Wednesday, February 27, 2008

Aussie DevCon Date Change

Due to circumstances totally beyond the organizers' control it has become necessary to move the DevCon dates to the following: Training - Monday 26th May to Thursday 29th May; DevCon Proper - Friday 30th May to Sunday 1st June.
Posted Wednesday, February 27, 2008

Clarion# Invoicing App Progressing

Randy Rogers notes that the Clarion# version of his invoicing application is now about 80% code complete. He is making the executable version available for those who are interested in seeing what can be done with the current incarnation of Clarion#.
Posted Wednesday, February 27, 2008

ABC Free Templates and Tools Updated

Recent changes to the ABC Free Templates and Tools include: Added Code template to generate templatized code for every field in a table; Added option to "derive procedure-level class from global object" for vsClipboardClass; Fixed call to %SVExpresionEditor; Created vsDiskFileIODriverClass; Rewrote vsDiskFileIOClass to use vsDiskFileIODriverClass, added FileToString and StringToFile methods; Changed name of vsSysShellOpClass to vsShellClass, added .Shell and .ShellEx methods to support calling ShellExecute; Standardized naming convention to use vTYPE_, and vEQ_; Fixed keyboard send class; SHIFT+HOME through SHIFT+END combinations were sending SHIFT+NUM7, etc.; Class objects now use a named instance rather than the "default" instance.

Posted Wednesday, February 27, 2008

Gitano Discontinuing Clarion Utilities

Gitano Software is discontinuing all Clarion specific utilities (this does not include gReg). The source code for the utilities will be available for a short period of time so registered users (or anyone interested) can continue updating the utility. Please note: Users who are within their 1 year of free upgrades will continue to receive the updates as they become available; New and upgrade purchases to the source code will include 30 days support.

Posted Wednesday, February 27, 2008

UltraTree, HyperBrowze, Up & Up Sale, Up to 50% Off

Enabling Simplicity is having a sale with up to 50% off regular prices. The more you buy, the more you save. The online order form calculates your discount and subtracts it from your order; quantities and previous purchases are taken into account. Sale ends March 17, 2008.

Posted Wednesday, February 27, 2008

DMC 1.1.0.6

DMC 1.1.0.6 is now available. Changes include: Fixed bug with XLS as source when REAL is found; Fixed bug with XLS as source and cloning name; Fixed bug with TPS as source and BLOB is found.

Posted Friday, February 15, 2008

Clarion Magazine

Clarion# Array Index And Class Instantiation Changes

by Dave Harms

Published 2008-02-27

The latest release of Clarion.NET, build 2957, introduced some important changes to the Clarion# language. These include a major alteration in how classes are declared, the removal of the smart assignment operator, and the move to zero-based indexes. I'll look at each in turn, but the one that seems to have caused the most confusion so far is the change in how classes are declared.

References and objects

As you read this article you may get a sense of déjà vu. After all, I argued the case for some of these changes last December, in Should Clarion# Drop Automatic Instantiation? Some of that information is repeated here but I've rewritten that material as an explanation of how Clarion# works, rather than how it might work.

In that article I complained about this sort of code:

```
s       ClaString(20)
t       &String

    CODE
  s = 'abc'
  t = 'def'
```

Now, you'll note that I've declared two variables, a ClaString (which is a Clarion# data type) and a String (which is a .NET data type). And the confusing bit for me was why I needed to use a & in front of String but not in front of ClaString. This was but one example of how Clarion# code, prior to build 2957, appeared bewildering. And the reason for that had to do with how Clarion (and until recently Clarion#) instantiates classes.

Automatic instantiation

If you have a bit of experience with object-oriented programming, you're probably familiar with the formal distinction between a *class* and an *object*. In general, the term class refers to the type (or definition, if you prefer) of an object; there can be one or more *instances* of the class, and each of those instances is an object. So class=definition, and instance=object.

In Clarion you have several ways of declaring a CLASS. Here's what I think of as a "normal" class. It's a type, which means it is not allocated any memory:

```
NameClass   CLASS,TYPE
Name          String(20),Private
SetName     Procedure(String newName)
GetName     Procedure,String
        END
```

```
NameClass.SetName       Procedure(String newName)
  code
  self.Name = NewName


NameClass.GetName       Procedure
  code
  return self.Name
```

The class definition begins with CLASS and ends with END. Following the class you have the source code for the methods (although these can also be in a separate source file).

Again, the TYPE attribute tells the compiler *not* to allocate memory to this class. To create an instance of the class you need to do something like this in your Clarion code:

```
MyClass &NameClass
    code
    MyClass &= new (NameClass)
```

MyClass is declared as a reference variable, as indicated by the & at the beginning of the label; that is, it's a null value until the NEW operator allocates memory for an instance of NameClass, and assigns a reference to that object to MyClass.

However, you can also declare NameClass without the TYPE attribute:

```
NameClass   CLASS
!... properties and methods
          END
! methods here
```

In this case you don't need to declare a MyClass reference variable; you can just go ahead and use NameClass directly (provided it's in scope, of course):

```
  code
  NameClass.SetName('Dave')
```

A class *without* the TYPE attribute is really two things: a class definition, and an automatically created instance of the class.

The value of automatic instantiation

Automatic instantiation in Clarion (not Clarion#) has a very important value: it makes memory management easier. Up until Clarion grew OO extensions, it was just about impossible to create a memory leak in a Clarion application. Oh, you could forget to empty a queue, but really there weren't any nasty memory-related problems waiting to happen because Clarion programmers didn't explicitly allocate memory.

And then came OOP, and with OOP came those two ways to declare classes. Conveniently, the default implementation (without the TYPE attribute) meant Clarion programmers could continue to code just as before, without any concern for cleaning up memory. As soon as a CLASS (declared without TYPE) went out of scope the runtime library freed up the associated memory. But to allow the full flexibility afforded by object-oriented programming, the compiler had to permit the runtime creation of objects, if the programmer so desired. That meant defining classes with TYPE and using NEW to allocate memory; naturally, if you allocated memory with NEW you had to free the memory

with DISPOSE, or your application would have a memory leak.

Reference variables

A reference variable is a variable that starts out as a null value and is assigned a reference to an object. In Clarion, and in Clarion# up until build 2957, all reference variables begin with the & character. Most importantly, the very concept of a reference variable, as distinct from a variable of the class' type, is a byproduct of Clarion's automatic instantiation.

In C#, which is the standard for all things .NET, classes are typically similar to a Clarion CLASS with the TYPE attribute, and objects are typed references, or what we know as reference variables. A reference can either be null, or it can be assigned an instance of a class; a TYPEd class cannot be a variable.

An unTYPEd Clarion CLASS, however, is somewhere in the middle. Not only is it automatically instantiated, but you can't use it as a reference variable. And the kicker is this is the default state for Clarion CLASSes; from a .NET perspective, it's a bit like building your application out of static classes.

Is automatic instantiation still needed?

The real question is whether automatic instantiation is still needed, and the short answer is no, it isn't. Making CLASSes default to automatic instantiation no doubt saved many of us from a host of memory leaks. In Clarion#, however, these kinds of memory leaks are no longer a concern. That's because the .NET platform employs something called automatic garbage collection.

Clarion developers have enjoyed automatic garbage collection from the beginning. For the longest time we didn't (couldn't!) allocate memory - the runtime did that for us - but we also never had to clean up memory. .NET takes that a step further; you can NEW objects to your heart's content, and when they are no longer referenced by any active code, the .NET garbage collector swoops in and scoops them up.

That bears repeating: in Clarion# you can NEW objects, and you *don't* have to DISPOSE them.

The one caveat is that you can't really predict when the garbage collection will occur, except that it will happen sometime after your class is no longer referenced by any running code.

The problem with references

Not only is the reasoning behind automatic instantiation no longer valid for .NET but defaulting to instantiating objects is exactly opposite to standard practice in .NET programming. And because most Clarion# applications are going to be littered with instances of .NET objects, in addition to Clarion# objects, having two opposite default states is a recipe for disaster.

SoftVelocity really had no option but to do away with automatic instantiation, both for the sake of compatiblity with .NET and for the sake of simplicity and ease of use.

The way it works now

Here's the NameClass rewritten for Clarion#. There are two changes. First, I've removed the TYPE attribute (the class is now automatically TYPEd, if you prefer to think of it that way), and second, I've changed STRING(20) to its .NET equivalent, ClaString(20) (I'll have more to say about String vs. ClaString a little later).

```
NameClass   CLASS
Name        ClaString(20),Private
SetName     Procedure(String newName)
GetName     Procedure,String
        END
```

```
NameClass.SetName       Procedure(String newName)
  code
  self.Name = NewName


NameClass.GetName       Procedure
  code
  return self.Name
```

To use the class in code, I declare it as follows:

```
MyName  NameClass
```

In Clarion that style would get me an instantiated NameClass; in Clarion# it now means an uninstantiated reference variable. Even though I have *not* used the & in front of NameClass I do need to instantiate NameClass before I use it:

```
code
MyName = new NameClass()
MyName.SetName('Dave')
message(MyName.GetName())
```

NOTE: Although I've removed the & from the MyName NameClass declaration, I could leave it in if I wished. The compiler will simply ignore the & character.

As I indicated earlier, no DISPOSE is needed. When this class goes out of scope, the garbage collector will, at some point, dispose of MyName. (You can, however, still DISPOSE of NEWed classes if you wish, and that can be important where you need to free up certain resources in a timely manner without waiting for the garbage collection system to do the job.)

Autodisposal

Build 2957 introduced a nifty new attribute called AUTODISPOSE. This attribute preserves the deterministic disposal that was a feature of automatically instantiated classes. If you don't want to wait on the garbage collector to clean up the memory allocated to your class and you don't want to explicitly call DISPOSE, just hang an ,AUTODISPOSE attribute on the end of your CLASS and as soon as the class goes out of scope DISPOSE will be called for you.

Strings are special

To paraphrase George Orwell's pigs in Animal Farm, all classes are equal, but some classes are more equal than others. And strings are a case in point. You may have noticed that although I had to instantiate NameClass with NEW, at no time did I ever instantiate the Name property.

In .NET, as in other OO languages (such as Java), strings are classes, so you would expect them to be subject to the same kind of requirements as other classes. That would mean an explicit call to create an instance of the Name variable, as in the following version of the class. I've set up a default constructor (NameClass.Construct) to create the ClaString instance:

```
NameClass   CLASS
```

```
Name          ClaString(20),Private
Construct     Procedure
SetName       Procedure(String newName)
GetName       Procedure,String
          END


NameClass.Construct procedure
   code
   self.Name = new ClaString(20)

NameClass.SetName      Procedure(String newName)
  code
  self.Name = NewName


NameClass.GetName      Procedure
  code
  return self.Name
```

In fact this version, which creates self.Name explicitly, will compile and run. But the constructor simply duplicates what the class was already doing. ClaString has a constructor which accepts a length parameter, and when the class is created, this constructor is automatically called. So there's no need to recreate the class once again, as in the above example.

Now let's say I change the variable from a ClaString(20) to a String, which is the .NET string type.

```
NameClass   CLASS
Name          String,Private
SetName       Procedure(String newName)
GetName       Procedure,String
          END


NameClass.SetName      Procedure(String newName)
  code
  self.Name = NewName


NameClass.GetName      Procedure
  code
  return self.Name
```

In this case the Name property is *not* instantiated when the class is created. But that's not really a problem, because of this line of code in the SetName method:

```
self.Name = NewName
```

You may be wondering why that code works. After all, self.Name is a null reference at this point. Won't assigning a value to a null reference cause an exception? Ordinarily, yes, it would, but as I said, strings are special. What really happens in this case is that instead of assigning NewName to self.Name, the compiler actually executes code that is the equivalent of this:

```
self.Name = new String(NewName)
```

In other words, assigning a string value to a String variable actually calls the String's constructor, passing in the new value.

Z has indicated that you can force the use of constructors at class instantiation by passing parameters or using an empty parameter list () in the declaration. For instance, to initialize a String variable to an empty string you would declare it this way:

```
s     String()
```

That doesn't seem to work in the current build, but I assume it will in the future.

Immutable strings

.NET strings, as defined by the System.String class, are immutable. That means that you can't actually change the value of a String; any attempt to do so actually results in a new String.

In the Clarion world we're used to strings as repositories for string data. Consider the following Clarion (not Clarion#) code:

```
MyString       STRING(20)
  CODE

  MyString = 'abc'

  MyString = 'def'
```

In this code there is only ever one instance of MyString, and it is assigned different values at different times.

Now consider this Clarion# code:

```
MyString       STRING
  CODE

  MyString = 'abc'

  MyString = 'def'
```

After the first line of code, MyString is a System.String instance containing the value 'abc'. After the second line of code, however, the first instance of MyString has been destroyed, and a new instance of MyString has been created.

Because System.String instances are immutable, there's no such thing as declaring a String with a predeterimined length; Strings are whatever length you make them. ClaStrings, on the other hand, act like the Clarion STRING of old.

To recap: because strings are such common data they're treated specially; you don't need to NEW them to use them as the appropriate constructor will be called for you, either at class instantiation or when you assign a string value.

Smart assignment operator

Since & is no longer needed when declaring class reference variables, the smart operator - written as := - is no longer

needed. The idea behind the smart operator is it you could use it wherever you would use either & or &= and it would figure out which was needed.

And by now you may be thinking that & and &= are gone as well. Actually that's not the case.

There are always exceptions

Although you no longer need to use & when declaring class reference variables and you can use = instead of &= when assigning a class instance (an object) to a class reference variable, there are still places where you need to use & and &=. Specifically, you use these when creating references to value types.

In .NET data can be of reference type or of value type. A reference type doesn't contain the actual data - it contains a reference to the data. Think of class instances (objects) being assigned to reference variables.

Value types, however, contain the actual data; you don't have to create and initialize a reference. Clarion value types include SHORT, LONG, DATE, GROUP, QUEUE, and FILE, as blogged by Bob Z, and presumably a few others such as BYTE, USHORT, ULONG, TIME, REAL, SREAL, BFLOAT8, DECIMAL, PDECIMAL, KEY, BLOB, and VIEW (although I haven't confirmed the latter list).

The trick is that if for some reason you need a reference to a value type, rather than the value type itself, you'll need to declare the reference variable with & and assign the reference with &= in the old Clarion style.

Array indexes

There's one other platform compatibility issue that's raised its head lately, and that's array indices. Clarion has always had 1-based arrays; that is, the first element in any array has an index of 1, followed by 2, etc. In .NET, however, arrays are zero-based.

Why was this change needed? Well, if you're mixing Clarion# and .NET objects, which is pretty common in the Clarion# code I've seen so far, you can easily have a situation where sometimes you need to be zero-based, and other times one-based. Here's an example:

```
a    String[]
  CODE
  a = new String[5]
  a[1] = 'abcdefg'
  message(a[1].Substring(1,3))
```

The variable a is an array of System.String objects, and because it's created in Clarion# it uses Clarion#'s indexing scheme. But the Substring method uses a zero-based index. Prior to build 2957 the output of this code would be

```
bcd
```

which is probably not what you would expect. You had to use this code:

```
System.Console.WriteLine(a[1].Substring(0,3))
```

to get the first three characters.

The following code, as of build 2957, uses zero-based indexing everywhere and displays the string "abc":

```
a    String[]

  CODE
```

```
a = new String[5]
a[0] = 'abcdefg'
message(a[0].Substring(0,3)
```

Here's another example. The following code creates a three element string array, and displays each element in a message box:

```
s           String[]
str         String
x           long

  code
  s = new String[] {'abc','def','ghi'}
  loop x = 0 to 2
    message(s[x])
  end
```

The alternative to indexes

You don't necessarily have to burden yourself with array indexes: instead, you can use the new ForEach statement, which lets you iterate through an array. Here's the previous string array example written with ForEach:

```
s           String[]
str         String
x           long

  CODE
  s = new String[] {'abc','def','ghi'}
  foreach str in s
    message(str)
  end
```

Summary

Here are the key points to remember when declaring classes and reference variables in Clarion#:

- The default for all Clarion# classes is *not* instantiated (the TYPE attribute is implicit)
- You no longer need & for class instance variables (although you can leave it in if you like, and the compiler will ignore it)
- You need to NEW any Clarion# classes you wish to use
- You do not need to NEW strings - they are allocated automatically either when the class is created or when a value is assigned to the string variable
- You do not need to DISPOSE your classes, although you can if you wish
- To force DISPOSE when a class goes out of scope use the ,AUTODISPOSE attribute

Zero-based indexes are a little easier to remember: just start at zero instead of one! And to avoid confusion, consider using the ForEach statement where possible.

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).

Reader Comments

Posted on Thursday, February 28, 2008 by Kevin Erskine

Dave;

Thank you for such great articles. There is so much to learn, remember and code. Your articles are pretty straight forward and clear in the language and examples you use. I know I will get caught with the Zero-base stuff, so you suggestion to use foreach instead will be a great way the ensure all code is reviewed as conversion begins.

Thanks again; Kevin Erskine

Posted on Thursday, February 28, 2008 by Stephen Ryan

Thanks Dave

greate article as Kevin has mentioned and its great to see it all clearly set out for everyone in plain simple way.

Easy to ready and that makes the learning new things a pleasure.

Posted on Friday, February 29, 2008 by Dave Harms

Thanks, guys!

Dave

Posted on Friday, February 29, 2008 by douglas johnson

While I can't think of a great alternative immediately, I'm sure my eyes & mind will not be the only ones to read the following STRING code as being "equally undifferent" as the RETURN code:
MyString STRING
MyString STRING()

RETURN(MyValue)
RETURN MyValue

Posted on Monday, March 03, 2008 by Dave Harms

Douglas,

Yep, no argument there.

Dave

Add a comment

Clarion Magazine

Creating A Drag & Drop Batch Compiler

by Richard Rose

Published 2008-02-29

Batch Compilers are cool utilities to have because they save the aggro of having to compile each application individually. I wanted one that was nice and quick to setup; all I wanted to do was highlight a number of apps in Windows explorer and drag them onto a batch compiler window.

In this article I'll show you how I created just such a batch compiler. But first off here's a quick operational guide, if that's all you want.

Using the batch compiler

The first step in using the batch compiler is to create a Project Name; right-click on the Projects list box and choose Insert (see Figure 1).

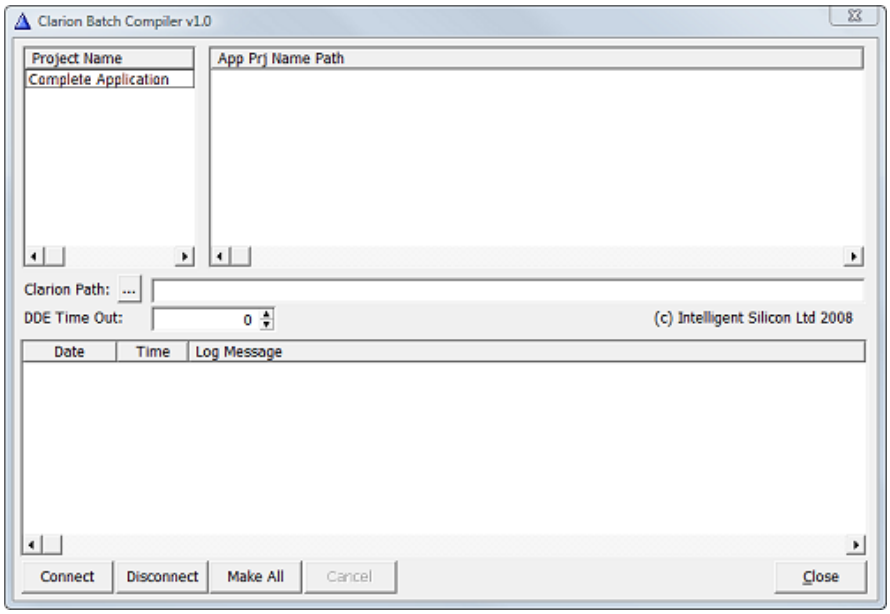

Figure 1. Adding a new project (view full size image**)**

You'll probably also want to specify a path to your Clarion EXE; if the path is invalid, the code will attempt to run C60EE.EXE and, if that fails, C60PE.EXE.

Now just drag and drop the APP files in the order you want them to be compiled.

Click Connect to load Clarion and establish a connection with the Clarion DDE server.

Now you're ready to compile. Just select the project and click Make All.

When you batch compile, the first file to be compiled is the one that's selected, so if you want to compile all then select the first (top record) in the list and the batch compiler will work its way down the files. If you want to resume partway (say after fixing a compile error) just select the app you want to use as the starting point, and then it will carry on down the list.

Other options:

- Click cancel if you want to stop compiling, clarion will finish off the current app.
- To clear the Log Messages just hold down the Del key as EIP works here too.

The nitty gritty

The first step in coding the batch compiler is to set up the (drag and) drop ID for File Explorer (not Internet Explorer). The drop ID is ~FILE, however this drop ID doesn't work on list boxes and some other controls as indicated in the help docs, so don't be misled. The only controls I have been able to drop ~FILE to are windows and Entry and Text controls. Okay, this is no big deal because the window will do just fine.

When you select one or more items from windows explorer and drag them to a window with the ~FILE DropID, you get the information in this format

```
"C:\MyFolder\MyFile.Ext, C:\MyOtherFolder\MyOtherFile.Ext"
```

In other words you get one string containing the drive, path and filename including extensions, and if multiple files are selected then they are separated by a comma. To see it for yourself just add a Message(DropID()) statement in the window's Event DropId event.

You'll need a local variable to store the dropped string; mine is Loc:FileDrop, defined as a STRING(8000); yes, it really is 8000 characters long. After all, I don't know how many files anyone might select in one go.

To parse the data, get the length of the string data and then loop through it starting at character position 1; break out of the loop when we find no more commas.

```
SetCursor(Cursor:Wait)
Loc:FileDrop = DROPID()
Loc:DropLen =  Len(Clip(Loc:FileDrop))
Loop Loc:DropPosition = 1 to Loc:DropLen
```

Next, search for the comma (if it exists) and store its position. If a comma is found its going to be above zero in value; extract the individual dropped file using SUB, and add it to the CompileApp file:

```
 Loc:CommaPos = INSTRING(',',Loc:FileDrop,1,Loc:DropPosition)
 Capp:CompileOrder = 1
 Capp:CompileProjectID = COM:RecordID
 Capp:AppPrjNamePath = |
  SUB(Loc:FileDrop,Loc:DropPosition,Loc:CommaPos-Loc:DropPosition)
 Access:CompileApp.TryInsert()
 IF Loc:CommaPos = 0
  Break
 END
```

After processing the file, assign the commapos to the drop position, reset the commapos and look for the next comma:

```
    Loc:DropPosition = Loc:CommaPos
    Loc:CommaPos = 0
  END
```

If there are no more commas the loop exits, and the following code resets the window:

```
  ThisWindow.Reset(1)
  SetCursor()
  Select(?AppList)
```

So that's parsed out the filenames from Windows Explorer and stored them in a table called CompileApp. I should point out that the order the files are listed are the order they compile in, so if you want a data DLL to compile first then drag that over first, then drag over the other DLLs, and finally the EXE.

Deja vue?

If you've looked at the source code you may have noticed a local procedure call RecordLogMessage, which has a string parameter. You can call RecordLogMessage anytime to store a message in the log file, which is displayed in the listbox at the bottom of the window. You will see lines of code like RecordLogMessage('Cancel Compiling request issued').

I could have made RecordLogMessage a separate procedure in the main app, because its used by both procedures Main and BatchCompile but I wanted to demonstrate how local procedures work and how they can be useful at cutting down on repetitive code. Here's the local MAP statement:

```
  MAP
    RecordLogMessage(String)
  END
```

And here's the code:

```
  RecordLogMessage   Procedure(String LogMessage)
    Code
    DLF:LogMessage  = Clip(LogMessage)
    DLF:Date        = Today()
    DLF:Time        = Clock()
    Access:DDELogFile.Insert
```

Using DDE

Right, if you didn't know, Clarion can be controlled by DDE. If you want a list of the commands that can be sent to it then look no further than "Using Clarion as a DDE Server" in the AdvancedProgrammingResources.pdf (thanks to Lee White for pointing me in this doc's direction).

In the source procedure called BatchCompile I have a loop which cycles through the APP files; this code issues a

DDEExecute request to Clarion, along with the app filename and path, instructing Clarion to generate and compile.

To use DDE you need to include a single line of code in the global map. That line is

```
INCLUDE('dde.clw')  !required whenever you need to use DDE Commands
```

Once that line is added you can use the DDE commands like DDEClient, DDEServer etc. as documented in the help PDF.

Connecting to Clarion

The first thing to do is to make sure Clarion is running and then connect to it using DDE. The line of code that establishes a connection is

```
Glo:Server=DDECLIENT('ClarionWin')
```

Glo:server just stores a number in a long, and it is a long number but never negative!

With this connection number you can then issue a command to Clarion, such as:

```
DDEEXECUTE(Glo:Server,'[ExecuteProject(' |
 & Clip(Capp:AppPrjNamePath) & ',0)]')
```

DDEExecute basically takes the Clarion IDE server number and a command, which in this case is encapsulated in square brackets. The command has this format:

```
ExecuteProject(â€˜MyPathAndFilename.app',0)
```

A value of 0 hides the app, and 1 shows it; see the aforementioned PDF for a better description of these parameters.

I check for any DDE errors with the CheckDDEError routine:

```
CheckDDEError   ROUTINE
  Loc:DDEErrorMsg = ''
  err# = ERRORCODE()
  IF err# > 600
    IF err# = 603
      DDEREAD(Glo:Server, DDE:Manual, 'GetErrorNum', Loc:DDEErrorNum)
      DDEREAD(Glo:Server, DDE:manual, 'GetErrorMsg', Loc:DDEErrorMsg)
      RecordLogMessage('DDE Error Num:' & Loc:DDEErrorNum & ', DDE Error Msg' & Loc:DDEErrorMsg)
    ELSIF err# = 605
      RecordLogMessage('DDE Timeout')
    ELSE
      RecordLogMessage('DDE Unexpected Error')
    END
  END
```

This code checks for errorcodes above 600, and depending on the errocode number further information can be got by sending a DDERead command. I don't know if there are other DDERead commands that can be sent to Clarion to interrogate it

further (if anyone knows please drop me a note at Richard@isv1.com)

The DDE:Manual equate basically tells Clarion you are issuing a one-off command to it; DDE:Auto creates a permanent or hot link to Clarion, whereby Clarion can send back any updates to the variable in the DDERead command. There are other events which you can see in the help.

After issuing the command to compile I close the DDELink to Clarion again because I don't want to tie up any additional resource or make programs hang or become sluggish. I don't know if there's anyway to get feed back on the status and progress of the ExecuteProject command; again let me know if there is.

The batch compiling thread

I start the BatchCompile procedure in a separate thread because I need to be able to cancel the batch job, and the BatchCompile procedure will be unresponsive while it's waiting for a DDE compile command to terminate.

```
START(BatchCompile,25000,COM:RecordID,Capp:RecordID)
```

You may not know that when a thread is started you can only pass in STRINGs or GROUPs (up to three); something that gets forgotten is that Clarion is very good at automatic type conversion. In this example I call the thread with two LONGs as parameters. Clarion takes the value from the longs and converts them to STRING parameters, and BatchCompile assigns them back to LONGs.

```
Capp:CompileProjectID   = pProjectId
Capp:RecordID           = pAppId !Start from where the hilight bar is
```

The bulk of BatchCompile is a loop that cycles through the CompileApp table until the project ID changes or there are no more records to cycle through, or until the Glo:CancelCompiling variable is set.

Filename tricks

One of the important bits in the loop is to be able to detect if Clarion previous crashed while compiling the application. I take the filename and path and change the last P in .APP to ~ so it becomes .AP~ . If you didn't know when you load an app file in Clarion, it copies the app file and calls the copy the same name but with an .AP~ extension. Assuming the app isn't loaded. there should be no .AP~ for the app file in question. Accordingly, if one doesn't exist I go on to compile it. Now if the .AP~ file is present I don't compile it, but record a message in the log indicating the presumably crashed .AP~ file was found.

Once the DDEExecute issued (meaning there was no .AP~ file) the code then loops until it finds the .AP~ file, because this indicates that Clarion has opened up the app as instructed in the DDEExecute request. Once the .AP~ is detected, the loop continues until this .AP~ file disappears, which usually means the compile completed and Clarion has closed the app. If the compile wasn't okay then a message will appear on screen which will possibly require user intervention (but don't quote me because there might be a way to interrogate Clarion under these circumstances).

A Sleep(1000) statement in the loop tells the thread process to go to sleep for 10 seconds, which is an efficient way of looping without hogging the CPU. Here's the declaration in the Global Map include:

```
MODULE('')
  SLEEP(LONG),PASCAL
END
```

Once the .AP~ file disappears the loop moves onto the next file and repeats everything over again.

I suppose I could have built the application with more intelligence for handling compile errors, but I'm working on the basis that the place you use a batch compiler isn't when you're coding and debugging, but when you want to assemble a multi-DLL application and bring everything up to date.

So there you go, you have an example of how to do a file drop from Windows Explorer, and you have a quick and dirty Clarion batch compiler that's easy to set up and tinker with.

If you have any changes or improvements please post a comment below or send them to me by email.

Download the source

Reader Comments

Add a comment

Clarion Magazine

Understanding Clarion# Strings: Revisited

by Dave Harms

Published 2008-02-29

Editor's note: This is an updated version of an earlier article. As of build 2957, Clarion# no longer automatically instantiates classes, which makes some of the information originally published obsolete.

Clarion.NET brings numerous changes to the Clarion language (now called Clarion#). I discussed a number of these changes in Clarion.NET Language Changes: What's Gone and Clarion.NET Language Changes: What's New and, more recently, in Clarion# Array Index And Class Instantiation Changes. In this article I'll take an in depth look at how strings change in Clarion#.

In most applications strings account for the majority of memory allocation, and that's particularly true for typical data-intensive Clarion applications. In the land of .NET, however, a string is something quite different from a "traditional" Clarion string. In this article I'll look at the differences between Clarion and .NET strings and discuss how those differences might impact your code.

Traditional Clarion strings

Here's the formal syntax for the STRING data type in Clarion:

```
                    | length |
label    STRING(    | string constant |)      [,DIM( )][,OVER( )] [,NAME( )] [,EXTERNAL] [,DLL] [,STATIC]
                    | picture |               [,THREAD] [,AUTO] [,PRIVATE] [,PROTECTED]
```

But that's pretty complicated. Most STRING declarations are simply a label and a length:

```
S       STRING(20)
```

The most important point is that STRING is a fixed length string. You specify how much space you need, and that memory is automatically allocated. You don't need to clean up this kind of string after you're done with it.

You also, however, have the option of creating strings dynamically, at runtime. In this case you declare the string as a reference variable without a specified length:

```
S       &STRING
```

Until you do something with this reference it has a null value; you have to create the string using the NEW operator:

```
S &= NEW(String(20))
```

And when you're done you have to dispose of the string this way:

```
DISPOSE(S)
```

Failure to dispose means the memory is not freed until the application terminates; this unfreed memory is called a memory leak, and memory leaks are Not A Good Thing.

Value and reference types

Before I get into .NET strings I want to explain a bit of terminology. In Clarion, and in Clarion# (as in .NET generally) there are two types of variables: value types and reference types. The code

```
Sval     STRING(30)
```

is an example of a Clarion value type. The variable is the 30 byte string. On the other hand, the code

```
Sref     &STRING
```

is an example of a Clarion reference type. Sref is not the string, it is a reference to the string. That's an important and subtle distinction.

Value string types can be accessed faster because the variable contains the string; reference types contain the address of the string, so there's an extra step involved in locating the actual string in memory. (I'm using the example of a string, but reference types can be just about any kind of data.)

The benefit of using a value type is speed; the benefit of a reference type is flexibility. Although most Clarion developers use value type strings, there are situations where using a reference type is an advantage, as when you'll be dealing with strings whose size you won't know until runtime.

.NET strings

.NET strings are quite a different animal from Clarion strings, although they bear some resemblance to string references. The .NET string datatype is System.String. Here's a System.String declaration in Clarion# (the USING SYSTEM directive is implicit, so the System. namespace doesn't have to be specified);

```
s            String
  code
  s = 'abcdef'
```

Strings in .NET are instances of the System.String class. That is, each string is an instance of a class, a.k.a. an object, with properties and methods. And System.String is a reference type, not a value type (although it's generally treated in code as a value type - more on that later).

The second important point is that .NET strings are fixed length and immutable. When you assign a value to a System. String, the string is exactly as long as the data. There's no padding, no empty space. When you assign a new value, or append a value to a string, the old string data is discarded and a new string is automatically created.

Along with a Length property, System.String has numerous methods including:

- Clone
- Compare
- CompareOrdinal
- CompareTo
- Concat
- Copy
- CopyTo
- EndsWith
- Equals
- Format

- GetEnumerator
- GetHashCode
- GetType
- GetTypeCode
- IndexOf
- IndexOfAny
- Insert
- Intern
- IsInterned
- Join
- LastIndexOf
- LastIndexOfAny
- PadLeft
- PadRight
- Remove
- Replace
- Split
- StartsWith
- Substring
- ToCharArray
- ToLower
- ToString
- ToUpper
- Trim
- TrimEnd
- TrimStart

As you can guess from that lengthy list, most of Clarion's string handling functions (SUB, UPPER, INSTRING etc) have direct equivalents in System.String methods. And since SUB, UPPER etc. all still exist in Clarion#, it also seems plausible that these methods could simply call System.String methods as needed. But there's at least one big problem with just using System.String in place of STRING in Clarion# code, and that has to do with STRING's fixed length. There are also some issues with string slicing and automatic type conversion.

Needed: A .NET string of arbitrary length

When you declare a STRING in Clarion you give it a length, but you don't do that with System.String. You can give a System.String an initial value if you want, but the length of a System.String is always the length of its data (sort of like a CSTRING).

SoftVelocity's solution is Clarion.ClaString, a completely new class which is evidently *not* derived from System.String. Here's a code snippet (note that the USING Clarion directive is implicit, just like the USING System directive):

```
s           ClaString(20)
 code
 s = 'abcdef' ! String is 20 characters long
```

Interestingly, if you leave off the length specification ClaString behaves just like System.String, and is only as long as the data it contains:

```
s          ClaString
  code
  s = 'abcdef' ! String is six characters long
```

ClaString's methods include:

- AssignString
- Clone
- Dispose
- Equals
- GetBufferSize
- GetBytes
- GetHashCode
- GetReferenceToSlice
- GetSlice
- GetType
- IsBytesDeposit
- RefRequal
- ReplaceSlice
- ToBoolean
- ToByte
- ToChar
- ToDateTime
- ToDecimal
- ToDouble
- ToInt16
- ToInt32
- ToInt64
- ToSByte
- ToSingle
- ToType
- ToUInt16
- ToUInt32
- ToUInt64

Properties include

- Kind
- Length
- RefType
- Value

You can see all of the To... methods that accommodate Clarion's automatic type conversion - similarly there are a bunch of constructors that make it possible to assign numeric values to strings.

The upshot of all of this is that you can still do string handling in Clarion# almost exactly as you do in Clarion, provided you use ClaString instead of System.String.

ClaString vs. String

Does all of this mean that you should always use ClaString in place of System.String? Probably not.

There are at least two issues here. One is that you'll have to change all of your STRING definitions to ClaString. Presumably a conversion tool could be created to take care of this task relatively painlessly.

> **NOTE:** You may be wondering why Clarion.ClaString isn't Clarion.String. The problem is name collisions. Every .NET application (at least, every one I can think of) needs classes declared in the System namespace; every Clarion# application that uses Clarion-specific functionality is going to need to use classes declared in the Clarion namespace as well as the System namespace. If you have identical labels in two namespaces, and you have USING directives for both those namespaces, the compiler is going to get confused. It won't know if String means System.String or Clarion.String, so you'll have to use the fully qualified name instead of the much shorter class name. And that's a pain.

The second issue is that ClaString is a completely different class from System.String, and if you write code that other .NET languages can use you'll have to stick to System.String anyway if you want those languages to call methods in your classes and pass or receive strings.

Finally, since ClaString is not derived from System.String, you have to use the Clarion RTL functions for string manipulation, which may make it more difficult to port sample code from, say, C#. Happily you can assign a String to a ClaString (and vice versa) so that isn't necessarily a huge problem, as long as your code makes it relatively clear which strings are .NET strings and which are Clarion strings.

Here's a little console application that illustrates some of the differences between ClaString and String manipulation:

```
s       ClaString(20)
t       String

    CODE
    s = 'abcdef'
    t = s
    s = sub(s,0,3)
    t = t.Substring(0,3)
    System.Console.WriteLine(s)
    System.Console.WriteLine(t)
    System.Console.ReadKey()
```

Note that I didn't use & in front of either the ClaString or the String declaration. In Clarion#, as of build 2957, all classes are implicitly TYPEd, and the & is not needed. Both s and t are in fact reference variables. And it may seem odd that as they are reference variables there's no place in my code where I explicitly NEW them.

Because strings are such a commonly-used data type they get special treatment, to the point where they're handled more like value types. You don't need to explicitly call a constructor when creating either a ClaString or a String; just assign a value, and the constructor will be called for you. For more information on strings and the recent changes to class instantiation see Clarion# Array Index And Class Instantiation Changes.

Back to the code: the ClaString s is assigned a value; then the System.String t is assigned a copy of that value. The Clarion SUB function extracts a substring and assigns it to s, just as the SubString method does.

But be careful. In Clarion, array indexes and other subscripts are 1-based; this applies to functions like SUB as well. The above code, using Clarion-style SUB indexes, will yield the following output:

```
bcd
```

```
abc
```

What you really want, now that Clarion# uses zero-based indexes, is s = sub(s,0,3) which yields the same result as t. SubString(0,3). That's a bit of a change for those of use used to 1-based indexes, but it does make for a much easier time when mixing Clarion# and .NET classes.

Array indexes are just one of the subtle ways Strings are different from ClaStrings. Another has to do with how references are assigned.

The following code creates two string reference variables, assigns the second to the first, and then changes the second:

```
s       ClaString
s2      ClaString


    CODE
    s = 'abcdef'
    s2 = s
    s2 = 'ghijkl'
    System.Console.WriteLine(s)
    System.Console.WriteLine(s2)
    System.Console.ReadKey(
```

The output is

```
abcdef
ghijkl
```

You might have expected that since s2 was assigned to s, that s2 was really a reference to s. But in fact that's not what happened. Instead s2 was simply created as a new ClaString with the value of s, so changing s2's value only changed s2 and did nothing to s.

As I said earlier, string classes behave differently than regular classes. Consider the following code:

```
MyClass          Class
str              ClaString
                 End

MyClassA         MyClass
MyClassB         MyClass


   CODE
   MyClassA = new MyClass()
   MyClassA.str = 'abcdef'
   MyClassB = MyClassA
   MyClassB.str = 'ghijkl'
      System.Console.WriteLine(MyClassA.str)
      System.Console.WriteLine(MyClassB.str)
      System.Console.ReadKey()
```

In this code I've created a class called MyClass, with a single string property. I have two reference variables of type

MyClass: MyClassA and MyClassB. I create a new instance of MyClass and assign it to MyClassA, and set MyClassA.str to 'abcdef'. Next I assign MyClassB to MyClassA and change the value of MyClassB.str. So what do you think will be the output? Do I have two str variables, or just one?

The answer is one, and the output is

```
ghijkl
ghijkl
```

As MyClass a "normal" (i.e. not a string) class, any variables of its type are reference variables which can be assigned directly to an instance of the class (via NEW) or indirectly via an assignment to another reference variable.

But how do you get a reference to a string, if the = operator just calls the constructor on a new string instance? By using ANY:

```
s              ClaString
s2              Any

    CODE
    s = 'abcdef'
    s2 &= s
    s2 = 'ghijkl'
    System.Console.WriteLine(s)
    System.Console.WriteLine(s2)
    System.Console.ReadKey()
```

The output of this code is

```
ghijkl
ghijkl
```

You must use &= to assign the reference to the ANY variable; if you use = you'll simply get a copy of the string.

This trick only works with ClaString; try it with String and you'll get a compile error. In fact, creating a direct reference to a String in .NET is problematic simply because Strings are immutable. Change the value of the referenced String and you've created a new String instance, and the previous instance, which is the one to which you have a reference, is discarded. You're better off using the indirect method of a reference to a class containing a String, as in the earlier example.

Summary

The new ClaString data type goes a long way toward bringing traditional Clarion string functionality to Clarion#. Still, there are a few points to keep in mind:

- ClaString references, like other class references, are now declared *without* the & character
- ClaString and String have implicit constructors; you don't need to NEW instances, you just assign values
- Use String for compatibility with other .NET languages
- Use ANY and &= to create a reference to a ClaString
- Direct references to String instances are problematic, because String is immutable
- You can always creat a class wrapper for Strings and/or ClaStrings and reference the class

---

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).

Reader Comments

Add a comment

Clarion Magazine

Getting Useful Information Out of SQL, Part 3

by Steven Parker

Published 2008-02-22

In the last action packed episode, I showed how Troy Sorzano created the dummy table trick to get the results of Prop:SQL queries into program variables and how subsequent Clarion developers expanded on the dummy table concept. The technique, however, only handled queries for totaling. That is, queries designed to return only one row.

Reports typically have multiple rows (or, more properly, an indeterminate number of rows whereas a query used to get totals always returns exactly one row). So, the question now is "How do I retrieve multiple rows from Prop:SQL?"

For example, I want a report of sales for each item:

```
dummy{Prop:SQL} = 'select PLU, ' &|
  'sum(AMT), '  &|
  'sum(Cost), ' &|
  'sum(AMT - Cost) as profit, ' &|
  'sum(QTY), ' &|
  'Count(Distinct ReceiptNumber) ' &|
  'from CUSPLU group by PLU;'
```

Remember that when Prop:SQL returns a data *set*, "you use NEXT(file) to retrieve the result set one row at a time, into the file's record buffer."

Obviously, I am going to need to use a loop of some sort. What is not obvious is that the loop inherent in a Report (or Process) will not work. More precisely, noting that the documentation says "you use NEXT(file)," it must be ThisReport.Next (or ThisProcess.Next) that doesn't work. Possibly the Next methods are designed to read physical files/tables and, as discussed, the physical file, not being used, is empty. (If you check the ProcessClass.Next in abreport.clw, you will find that the class is indeed designed around using physical files and tables. Remember, no CREATE attribute, no I/O, the physical file is empty. Therefore, the report engine will never find a record to print.)

The proof of this is found in creating a report with dummy as its primary table. Issue the Prop:SQL in an appropriate embed. Run the report and it will stop with a message at Take.NoRecords.

Why, then, not "just" write the buffer to the physical table? The simple answer is that the file has no create attribute, no keys, no way of filtering. What if multiple users are running reports and hitting dummy? What about records already in dummy? Thinking it through, using the actual file in a multi-user or multi-threaded environment entails a significant amount of work to be effective.

Using a dummy table

It is possible to use a dummy table in a report even though the dummy table does not physically exist. However, prior to Clarion 6, it is somewhat more difficult. This is because one of the major, and little noticed, enhancements in C6 is that a report source no longer *must* be a dictionary file. Even though you must still name a file in the file schematic, that name can be ignored -the template does, so you can too. I'll describe the process for C6.x.

From the report's main Property Worksheet, press Report Properties and select the Data Source drop list:

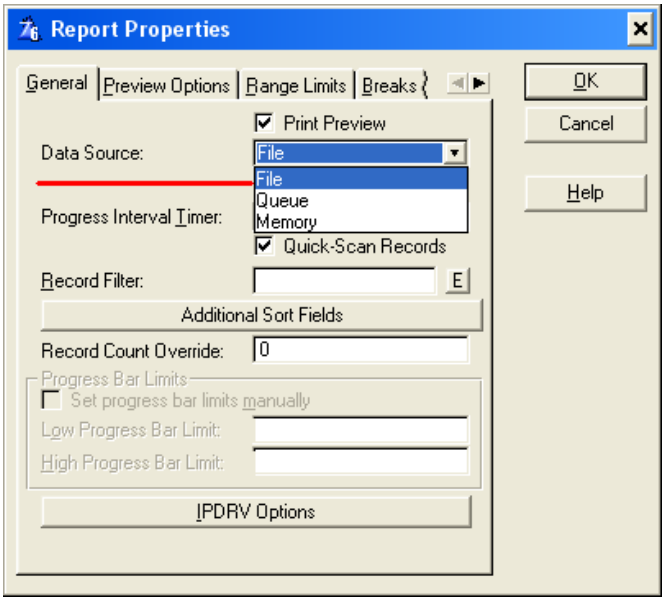

Figure 1. Alternate data sources for C6 reports

As Figure 1 shows, you can set the Data Source to "Memory" and not have to worry about ThisReport.Next accessing the physical records. (In earlier Clarion versions, standard sequential access must be short circuited. Manually.)

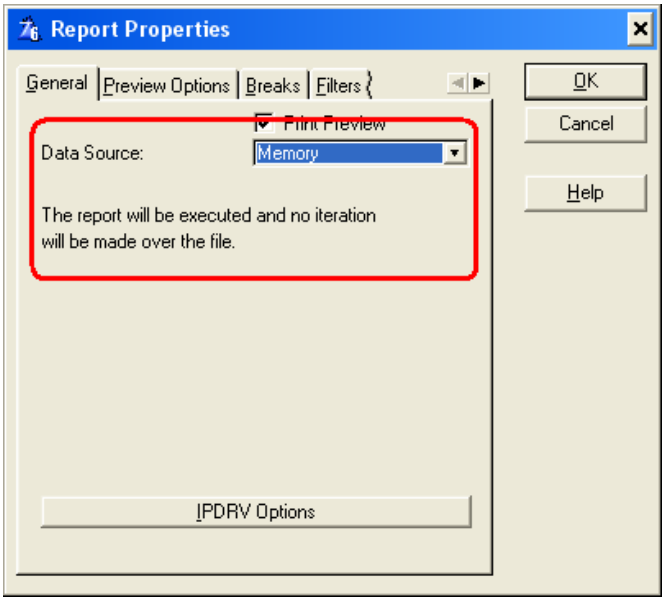

Figure 2. Template prompt for Data Source = "Memory"

Late in INIT, issue the Prop:SQL. Remember that because there has been no Next(), nothing has been loaded into dummy's buffer (and Records(dummy) will always return zero). In TakeRecord, After Parent call (this is immediately before the default Print(RPT:Detail) but inside a loop),

```
Next(dummy)
If ErrorCode()
  Return Level:Benign
End
```

```
Print(RPT:Detail)
Return Level:Notify
```

The key here is the Return Level:Notify statement. Because this embed is before the built in Print, this Return by-passes the normal processing sequence of the template. This approach is required for all Clarion versions. (I have tried letting the default Print statement do the printing and could not make it work.)

However, in by-passing the standard template actions, I also lose the built in updating of the progress bar. I need to do that myself (see A Progress Bar For Multiple Processes for how to do this).

I also need to do any break handling myself. This subject has been well covered in previous Clarion Magazine articles (e. g., Completely Dynamic Report Orders and Breaks Part 1; see the sidebar for more references).

All in all, this seems like a substantial amount of work to go through (given that I am on C6.3 and not an earlier version). There is almost no point in using a template report if I have to do the progress bar and break handling myself for even the simplest of reports.

But, it does work.

Using a Real File

Of course, using a real file (an In-Memory Database Driver file seems an ideal candidate, not to be confused with the "Memory" setting just described) could eliminate a number of the difficulties of using the dummy table directly. I do not think there is a good reason not to use a temporary TPS or IMDD file. Such a file could contain as many Strings or CStrings as the largest query I might issue, like dummy (as discussed previously). This temporary file could contain a mix of strings, numerics and decimals for me to load, selectively. Then,

```
dummy{Prop:SQL} = 'select PLU, ' &|
  'sum(AMT), '  &|
  'sum(Cost), ' &|
  'sum(AMT - Cost) as profit, ' &|
  'sum(QTY), ' &|
  'Count(Distinct ReceiptNumber) ' &|
  'from CUSPLU group by PLU;'
Create(TEMPFile)
Open(TEMPFile)
Set(dummy)
Loop
  Next(dummy)
  If ErrorCode()
    Break
  End
  TMP:Field1 = DUM:Field1
  TMP:Field2 = DUM:Field2
  ! etc
  Add(TEMPfile)
End
Close(TEMPFile)
```

Then, the temporary file can be the primary file for the report. The only issue would be formatting the strings in the report formatter to reflect the actual data type.

When the report is done, I can close and remove the temporary file. In the meantime, the progress bar is handled by the template, subtotaling is handled and all I need to deal with is group breaks. And, if I use SetOrder, the standard break handling will handle breaks correctly.

Printing from a queue

Clarion 6 makes printing from a queue easy. The only thing I have found different when using a queue is that ValidateRecord does not seem to be called when the data source is a queue. I can live with that.

To me, a major advantage of using a queue is that I can declare it locally in either a data embed or by using the Data button on the procedure's Properties worksheet. (The advantage of using the Data button is that I can use the "E" button to select the queue from a list; if declared in an embed, I have to type the label of the queue. Also, declaring the queue in the Data button allows me to use my queue elements directly in the report formatter). Declared locally, I have no concern about other users or other procedures accessing the data and, possibly, changing it. Also, I can format the queue elements correctly before loading it.

One other thing I can do is apply a manual filter that, for whatever reason, I could not do in my SQL query. In fact, because I am loading the queue using standard coding techniques, I can bring everything I know to bear on loading and processing the queue.

```
Free(ReportQ)
Clear(ReportQ)
Set(dummy)
Loop
  Next(dummy)
  If ErrorCode()
    Break
  End
  If DUM:Field47 < LOC:FloorValue ! additional filter
    Cycle
  End
  RQ:Field1 = DUM:Field1
  RQ:Field2 = DUM:Field9 ! no need to be "sequential"
  ! etc
  Add(ReportQ)
  LOC:GrandTotal += RQ:Field16  ! accumulate totals
End
```

If I sort the queue on the elements I want, I do not have to check breaks manually (in this case, assuming that I declared the queue somewhere the IDE can see it, the built in break manager will respect queue elements). I like this technique and it is fast, very fast, even for large data sets. Queues, like temporary files, significantly lessen the expertise needed to deal with SQL databases (read "shorten the learning curve").

Using The ADO Classes (without using ADO)

"It sure would be handy if Clarion had a way to run an SQL query and just get back a result.... The problem is that in Clarion I need a view to project" to get the desired datum, mused Tom Ruby. Then he noticed that SoftVelocity, in the 6.x cycle, "gave us a class they called cCwADO which handles this kind of situation perfectly" (see Querying SQL Data).

This class allows retrieving the results of a query, whether a single row total or a multiple row set, without a dummy table, view or temporary table. In the case of a single row, a total for example, no post-query processing is needed. None.

The cCwADO class does require:

(1) including the class:

```
include('cwado.inc'),Once
```

I put this in Global Data, After File Declarations.

(2) instantiating two classes (I use local data embeds):

```
ComIniter cCOMIniter  ! COM subsystem
DataSet   cCWAdo! ADO classes
```

(I recall trying to instantiate the COM subsystem in a Global Data embed without success. But that may well have been something I did wrong. It's worth trying.)

(3) and several new pragmas in the project settings:

```
_COMLinkMode_
_COMDLLMode_=>off
_ADOLinkMode_
_ADODLLMode_=>off
_svLinkMode_
_svDLLMode_=>off
_ADOMPRLinkMode_
_ADOMPRDLLMode_=>off
_SVDllMode_=>0
_SVLinkMode_=>1
```

(4) The class requires a connection string (Before Open Files):

```
DataSet.SetConnection(GLO:CMSGLOBALDNS)
```

Tom notes that the connection string required here is *not* the same connection string used in the owner string in the dictionary. It looks like this:

```
Provider=SQLOLEDB.1;
  Persist Security Info=False;
  User ID=sa;
  PASSWORD=sa;
  Initial Catalog=<dataBase>;
  Data Source=<serverName>
```

In other words, this class seems to target ADO data sources (i.e., MS SQL). However, I used my dictionary connection string quite successfully with MySQL data bases. The only issue I ran into is that Date columns (MySQL supports Date data types, MS SQL does not, supporting only DateTime types) return without a value.

The heavy lifting is done when I tell the class where and how to assign my data. Read that again: the developer determines and assigns the data. It does require that each datum in the query have an alias and that there be a real (i.e., Clarion) variable for each datum (yes, even an implicit variable!) but this sounds like standard programming practice. Generically:

```
<myLocalClass>.AddFieldsInfo(" , ' <queryAlias> ', <variable> , 0)
```

Using my PLU sales total example:

```
'select sum(AMT) from CUSPLU;'


DataSet.AddFieldsInfo('', 'total',LOC:TotalSales,0)
```

And I can tell the class to run my query:

```
RetVal = DataSet.ExecuteQuery('select sum(AMT) ' |
'as total from CUSPLU;')
```

If RetVal is 1, the query succeeded (returned at least one row). Otherwise RetVal is zero. So,

```
If RetVal
 ! LOC:TotalSales already contains my total
End
```

Magic! I'm done because the AddFieldsInfo statement mapped the query column to a datum of *my* creation.

Return multiple columns? No problem (see Figure 3).

```
DataSet.AddFieldsInfo('','LossYear',LOS:LossYear,0)
DataSet.AddFieldsInfo('','LossState',LOS:State,0)
DataSet.AddFieldsInfo('','Cov',LOS:ResType,0)
DataSet.AddFieldsInfo('','Amt',LOS:Amount,0)
DataSet.AddFieldsInfo('','Count',LOS:ClaimCNT,0)

ExecString = 'SELECT Year(claim.lossdate) as LossYear, '   &|
             'tmppolicy.state as LossState, '              &|
             'salvagetype as Cov, '                        &|
             'sum(salvagerecovery) as Amt, '               &|
             'count(claim.claimno) as Count '              &|
             'FROM `salvtbl` '                             &|
             'inner join `claim` using (claimno) '         &|
             'inner join `tmppolicy` on (tmppolicy.sid = claim.poli
             'where salvtbl.updateddate >= "' & Format(GLO:BeginDat
             'and salvtbl.updateddate <= "' & Format(GLO:EndDate,@d
             'and claim.lossdate is not null and salvagerecovery !=
             'group by lossstate,LossYear,salvagetype'
```

Figure 3. Setting up and returning multiple columns (view full size image**)**

Execute AddFieldsInfo for each variable before ExecuteQuery.

Expecting an indeterminate number of rows, as in a report? With a local queue as the report source, set queue elements as the assignment target in AddFieldsInfo, as shown in Figure 3, above.

Then,

```
RetVal = DataSet.ExecuteQuery(ExecString)
 If RetVal <> 0    ! query successful
  Loop
   If ~LOS:LossYear  ! additional filter
    Cycle
   End
   Add(LossQ)
```

```
    If Not DataSet.Next()
      Break
    End
  End
End
DataSet.Close()
```

Note that I form my SQL query as a string and pass that string to ExecuteQuery. This is for readability. It also allows me to cut and paste my successful query from my SQL environment and add only Clarion's string delimiters and continuation marks.

Because all of the assignments of SQL aliases to Clarion variables are already specified in AddFieldsInfo statements, a simple loop is all that is needed to get the returned data set into a queue (or file or table or... whatever I want) which is, and this is the point of the exercise, entirely under my control and completely accessible to me.

Oh, yes, notice the last line. Close the set when done with it. Does that mean that the set can be reopened and another query issued?

Yes. In fact the query shown in Figure 3 is one of seven in a single procedure (and I used to be worried about *two* passes!). The only caveat that I ran into (and, perhaps, that was because I was not querying an ADO data source) was that I could not "reuse" an alias. So as lossyear in one query had to become as **x**lossyear in the next, and so on.

I need to reiterate the fact that the cwADO class appears to be specifically designed for recent MS SQL implementations and ADO data sources. I have used it successfully with MySQL though, as mentioned, date fields were unreliable (in those cases, I also returned the row's unique identifier and, in my queue loading loop, got the underlying row and read/deformatted the date). So, if using this technique with non-ADO data sources, test carefully.

Summary

Well, there it is. I *can* get the result of SQL queries into my program's variables. Whether I expect a single row returned, as in getting totals, or an unknown number of row, as reports, I can do it.

Not only can I do it, I have a choice of ways. In fact, there's even one more technique: after this series was submitted but before you got to see it, Robert Johnson gave us yet another way of getting reports out of SQL queries in Create a Report using MS SQL Views.

Yes, my personal preference, at least for reports, is a local queue. But Tom Ruby's use of the ADO class has low overhead and is fast; for ADO data sources, it's a winner. For non-ADO data sources it also works but may not handle all data types correctly. For single row returns, a local dummy file seems the way to go.

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.

Reader Comments

Posted on Friday, February 22, 2008 by Stephen Ryan

Does the use of the new clarion 6.2 command, SQLTurbo solve some of these questions?

Dlib also provides a solution in that a SQL file can be access with a temporary file with no fields declared.

Posted on Friday, February 22, 2008 by Stephen Ryan

Does the use of the new clarion 6.2 command, SQLTurbo solve some of these questions?

Dlib also provides a solution in that a SQL file can be access with a temporary file with no fields declared.

Posted on Friday, February 22, 2008 by Steven Parker

I do not see how TurboSQL would help (and, IIRC, not all SQLs support it). It just doesn't check columns.

As to DLIB, I expect to see the article.

Soon.

<G>

Posted on Saturday, February 23, 2008 by S Jayashankar

Hi Steve P,

The /TURBOSQL driver string is ideal for these dummy tables and it works on all SQL drivers EXCEPT the Oracle Native Driver. It DOES work on Oracle databases using the ODBC driver though. An ideal continuation of this series would be to explain how to execute SQL queries in a generic way i.e. result columns/ data unknown until runtime using the Windows ODBC32 library or DFD from SV. Just a thought - not a comment :-).

Shankar

Posted on Thursday, February 28, 2008 by Julio César Pedroso

Yes. cwAdo works fine, but i need to know: how to manipulate with this?
I use PostgreSQL trough ODBC.

Posted on Thursday, February 28, 2008 by Julio César Pedroso

Sorry. Replay...
Yes. cwAdo works fine, but i need to know: how to manipulate date fields with this?
I use PostgreSQL trough ODBC.

Posted on Saturday, March 01, 2008 by Gordon Holfelder

Hi Steve-

I use a similar technique using the dynamic file driver. Though there is an extra cost for the driver, for me it's closer to how regular clarion tables are processed. Your code, for example would be:

```
curTable          DynFile
curFile           &FILE
curRecord         &GROUP

 CODE
 !---------------------------------------
 ! Open and initialize SQL
 !---------------------------------------
 curTable.UnfixFormat()
 curTable.ResetAll()
 curTable.SetDriver('ODBC')
```

```
  curTable.SetOwner(SELF._ConnectionString.Get())
  curTable.CreateFromSQL( |
    'select PLU, ' &|
    'sum(AMT), '   &|
    'sum(Cost), ' &|
    'sum(AMT - Cost) as profit, ' &|
    'sum(QTY), ' &|
    'Count(Distinct ReceiptNumber) ' &|
    'from CUSPLU group by PLU;')
  curFile          &= curTable.GetFileRef()
  curRecord        &= curFile{PROP:Record}
  OPEN(curFile)
  !----------------------------------------
  ! Process database values
  !----------------------------------------
  LOOP
    NEXT(curFile)
    IF ERRORCODE() = 33
      BREAK
    ELSIF ERRORCODE()
      MESSAGE('Error processing select ' & ERRORCODE())
      BREAK
    END
    !----------------------------------------
    ! Assign to internal queue or whatever
    !----------------------------------------
    q.Value1 = WHAT(curRecord, 1)
.... more code ....
  END
  !----------------------------------------
  ! Close and exit
  !----------------------------------------
  CLOSE(curFile)
```

Posted on Saturday, March 01, 2008 by Steven Parker

Gordon,

You and Shankar have a couple of interesting articles. I look forward to reading them.

Posted on Monday, March 03, 2008 by Sed Mayne

Hi Steve,

What do I need to change to make the ADO example work from a dll in my multi-dll app.

Executing DataSet.ExecuteQuery(LOC:ExecString) gives me an immediate and silent GPF. The program just vanishes.

Have you doubled-up on some of the pragmas
    is
svLinkMode
    different to
SVLinkMode=>1
    for example?

Regards
  Sed Mayne

Posted on Tuesday, March 04, 2008 by Sed Mayne

Hi Steve

Solved it. No changes required for dll. My error. Bad query - all working now.

Not a lot of error checking in the ADO classes ...

Sed

Add a comment

Clarion Magazine

Getting Useful Information Out of SQL, Part 2

by Steven Parker

Published 2008-02-21

In the previous installment I traveled the road from being unimpressed with SQL to embracing it. Except, while I could do very sophisticated queries, I could not get the information into my programs.

The initial insight came from a careful reading of the documentation on Prop:SQL:

> You can use Clarion's property syntax (PROP:SQL) to execute SQL statements in your program code by using PROP:SQL and naming the FILE or imported SQL VIEW in the data dictionary as the target within the normal execution of your program.

This tells me that Prop:SQL is how one executes SQL queries against the back end. But I already knew this from lurking in the newsgroups. Note, however, that there is no requirement that the target file/view be the same as the tables named in the SQL statement(s).

More importantly, it tells me why my initial attempt to get a query's result into my program's variable

```
myLong{Prop:SQL} = 'select sum(AMT) from CUSPLU;'
```

failed. myLong is not a file or view structure, neither is it an entity from my dictionary.

The dummy table

So, Troy Sorzano, who first posted a solution to this problem (at least, it is my recollection that it was Troy), created a table in his dictionary that was used only for the purpose of receiving the results from SQL executed by Prop:SQL. He called it a "dummy table" and an exemplar might look something like this:

```
Dummy FILE,DRIVER('ODBC'),OWNER(GLO:ConnectString),|
            NAME('Dummy'),PRE(DUM),BINDABLE,THREAD
Record      RECORD,PRE()
Field1       CString(256)
          END
```

Note my use of a CString as the datum. Relying on Clarion's automatic data type conversion, any datum received will be readable. A String would work just as well.

Now I can issue SQL statements against a dictionary entity:

```
dummy{Prop:SQL} = 'select sum(AMT) from CUSPLU;'
```

This works fine; I don't get a GPF (a GPF is Clarion's way of saying that something is wrong with my MySQL query; MS SQL may return an ErrorCode 90). The next issue is getting the data into a variable to which I have programmatic access.

The documentation on Prop:SQL continues:

> If you issue an SQL statement that returns a result set (such as an SQL SELECT statement), you use NEXT(file) to retrieve the result set one row at a time, into the file's record buffer. The FILE declaration receiving the result set must have the

> same number of fields as the SQL SELECT statement will return. If the Clarion ERRORCODE procedure returns 90, the FILEERRORCODE() and FILEERROR() functions return any error code and error message set by the back-end SQL server.

The documentation in 5.5 reads "you *must* use NEXT(file)" (emphasis added). (Also see Showing ABC Errors for how to decipher ErrorCode 90.)

This implies that:

```
Open(Dummy)
dummy{Prop:SQL} = 'select sum(AMT) from CUSPLU;'
Next(Dummy)
```

will fill DUM:Field1 with the total PLU sales. And, indeed, this is the case. (I must remember to ensure that the table is eventually closed also; listing it in a procedure's file schematic under Other will ensure that appropriate Open and Close commands are issued.)

What if I want to retrieve multiple pieces of information? For example, I want the total sales, total cost, total profit, quantity and receipts:

```
select sum(AMT),
  sum(Cost),
  sum(AMT - Cost) as profit,
  sum(QTY),
  Count(Distinct ReceiptNumber)
  from CUSPLU;
```

In this case, my dummy table will need to have five fields.

This led many Clarion developers to realize that the documentation on Prop:SQL might not be strictly correct. Specifically, the statement "The FILE declaration receiving the result set must have the same number of fields" is not accurate.

The file declaration needs to contain *at least as many* fields as the number of fields returned. It can contain more fields than Prop:SQL returns. It cannot contain fewer (if it does, using MySQL, the app will GPF).

> **Note:** This statement, that a Clarion data declaration can contain a different number of fields than the SQL table has columns, is a general rule in Clarion. The only restriction is that a Clarion procedure cannot retrieve more columns than the data declaration has fields or columns it does not know about. In practice, this means that either the dictionary must be kept, approximately, in sync with the back end layout or the columns to be retrieved from the back end must be explicitly named in a Prop:SQL statement.

This caused people to include dummy tables in their dictionaries containing as many fields as the biggest SQL statement anticipated might return. See, for example, Dan Pressnell's series of Better SQL articles at Icetips contains a very complete and enlightening implementation of this approach.

It turns out that there is another minor mistake in the documentation. The statement "in the data dictionary" is also not strictly true.

Scott Ferret, TopSpeed's and SoftVelocity's driver guru, posted the following in March 2000. It is worth the time to study:

> An industrial myth has evolved over the last few years in the Clarion/SQL community regarding the need of a temporary table *on the server* so that you can issue SELECT statements from anywhere and have something to put in it....
>
> I believe that the reason people believe that they have to have this temp table on the server is they look at a Clarion FILE definition and believe that there must be an equivalent TABLE to match it on the server. *You do not need this.*

What the Clarion system requires is that the fields in the FILE definition match some fields in some table on the server. This is so it can get field type information from the server. The most important thing to realise is that the fields on the server do not have to be different fields. Here is a Clarion FILE definition that works with SQLAnywhere with a user who has sufficient priority to see the system tables. The idea can be adapted to work with any tables that you [k]now that your program will be able to see.

```
MyTempTable FILE,DRIVER('SQLAnywhere'),NAME('SysColumns')
Record  RECORD
ALongField    LONG,NAME('Length')
AShortField   USHORT, NAME('SysLength')
AStringField  STRING(100),NAME('cname')
LongField2    LONG,NAME('Length')
StrFld2       STRING(20),NAME('cname')
CStrFld       CSTRING(50),NAME('cname')
          END
        END
```

The important things to notice are:

1) The file's label has nothing to do with the name of the table on the server

2) The field's label has nothing to do with the name of the column on the server

3) You can use the NAME attribute on a field to have multiple fields point at the same column

4) You should *try* and use the same data types as the ones on the backend. But it is not necessary.

(Emphasis added.)

The points worth reiterating are:

- The dummy table does not have to be declared in the dictionary (it can be declared in a procedure's local data embed or anywhere else in the app that is visible to the Prop:SQL statement).
- The field data types do not need to match the returned data types (within the constraints of the automatic data conversions).
- The labels (table and field) do not need to match (anything).

SuRF does not explicitly mention one more important thing: the target file fields are filled in the order data is referenced in the Select statement but it is very important to remember this. Also, the dummy table does not have to be a table (file) at all. It can be a view.

As Geoff Bomford notes, in a newsgroup posting:

Use a Clarion View.

```
MyView  VIEW(AnyTable)
          PROJECT(AT:ALong)
        END
OPEN(MyView)
MYView{Prop:SQL} = What you said
Next(MyView)
! AT:Long now has your returned value
CLOSE(MyView)
```

But wait!

There are several things in the above examples that strike me as a bit ... odd.

First. The dummy table does not have the Create attribute.

```
Dummy FILE,DRIVER('ODBC'),OWNER(GLO:ConnectString),|
    NAME('Dummy'),PRE(DUM),BINDABLE,THREAD
```

There is a widely held belief in the Clarion community that one must never allow a Clarion program to create a SQL table.

The reason for this is that when Clarion creates a SQL table, it is unable to create all the attributes required. Specifically, Clarion is unable to create some important attributes on the primary key. Scott Ferret, again, in a news group posting in June 1998, states:

> If you want to create primary key constraints you need to use PROP:SQL to do the create. Note that you should always use PROP:SQL to do a create as the Clarion CREATE statement cannot handle all the features of the SQL CREATE TABLE statement. Eg [sic.] constraints on fields and foreign keys.

However, for the simple purpose of having a file to receive data, CREATE is irrelevant and unimportant (note that the declaration for dummy doesn't have any keys, much less a primary key, therefore, there can be no constraints). What is important is that when Clarion developers talk about the dummy table technique, they never talk about using the actual, physical table.

Okay, I could create a script to create the dummy table (the SQL UIs I have used can create a text file containing the necessary statements). Then,

```
Dummy{Prop:SQL} = 'createScript.sql'
```

will create the file, on the server, for me. If I really thought I needed a physical file, despite SuRF's disclaimer.

Second, the suggested code:

```
Open(Dummy)
dummy{Prop:SQL} = 'select sum(AMT) from CUSPLU;'
Next(Dummy)
```

doesn't write anything to the file. There is no Add or Put. There is no file I/O.

In fact, and this took me a while to grasp, the physical file/table is *never* actually used. This is the hard part to comprehend, to get your head around, but it is a singular and important fact. I repeat, the actual table is not used. The physical table is irrelevant.

The only thing that is actually used is the data declaration and, specifically, the Record structure. Opening a file or table creates a file buffer:

> At run-time, the RECORD structure is assigned memory for a data buffer where records from the disk file may be processed by executable statements. This record buffer is always allocated static memory on the heap, even if the FILE is declared in a local data section.

and it is the buffer that *is* used. To reiterate the Prop:SQL documentation, Prop:SQL uses the buffer: you "retrieve the result set ... *into the file's record buffer*" (emphasis added). Prop:SQL does not use the table.

Summary

The above example involves returning and reading only a single datum or, more properly, a single row. So far the dummy table technique handles cases where I want a total or a record count, or both. It does not handle reports.

Next time: reports and a choice of ways to get query data sets into my own variables.

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.

Reader Comments

Add a comment

Clarion Magazine

Getting Useful Information Out of SQL, Part 1

by Steven Parker

Published 2008-02-15

The very first database program I used was called CONDOR3. I ran it on CP/M and later on DOS 3.0. It used ASCII files for storage and is widely considered the first RDBMS (Relational DataBase Management System) for microcomputers. In recognition of that fact, a copy resides in the Smithsonian Institution, so labeled.

CONDOR3's syntax was clearly SQL. I retrieved data with Select and filtered with Where. CONDOR3 featured Join and Project statements and a number of other features I've long forgotten.

So I naturally approach SQL as something that can, should and does return to me information that I can use and returns it in a directly useable form. CONDOR3, when asked to return information, displayed on screen in full record browse mode. When asked for a report, it printed a report. In fact, like modern SQL engines, CONDOR was actually a scripting language (except that my CONDOR scripts could also create a user interface).

In the early 90's I did some work with Oracle. After spending a few days figuring out various ways of connecting my Clarion application to the Oracle database, I was left singularly unimpressed. After attending several days of an introductory Oracle class, I was impressed by the amount of work necessary to keep an Oracle database running. I considered that simply excessive.

MS SQL 6.5 and 7 left me equally unimpressed. I did not see much in the way of benefit to me as a developer in exchange for the complexity (okay, I did like having a full featured data base manager, a UI sorely lacking in Clarion). Connection strings, DSNs, date/time fields, etc., etc. didn't seem to buy me anything in return. (Connecting to SQL back ends was and remains a very frequent topic on the SoftVelocity news groups. See http://www.connectionstrings.com/ to get an appreciation of the complexity of having to configure connection strings which allow end users to connect to a SQL database without having to create a Data Source for each and every table, for that is the alternative to a DSN-less connection.)

Yes, I became aware that batch updates could be easier with modern SQL engines — the engine builds in a degree of intelligence. For example, to reset inventory using TPS files at the beginning of a new fiscal year, I would use this code:

```
Loop Until Access:Inventory.Next()
  INV:YTDSold = 0
  INV:YTDReceived = 0
  INV:VNDYTDRec[1] = 0
  INV:VNDYTDRec[2] = 0
  INV:VNDYTDRec[3] = 0
  INV:VNDYTDRec[4] = 0
  INV:VNDYTDRec[5] = 0
  Put(INVENTORY)
End
```

In SQL, no loop is necessary and my code would look like this:

```
inventory{Prop:SQL} = 'update inventory set ' &|
```

```
    ' YTDSold = 0 '     &|
' YTDReceived = 0 ' &|
' VNDYTDRec_1 = 0 ' &|
' VNDYTDRec_2 = 0 ' &|
' VNDYTDRec_3 = 0 ' &|
' VNDYTDRec_4 = 0 ' &|
' VNDYTDRec_5 = 0;'
```

(The data declaration has been adjusted: "flat" files support arrays, SQL, by and large, does not.)

In the SQL version of resetting inventory, a single statement is sent to the back end and the back end handles the update (the "intelligence" I referred to earlier). The TPS version of inventory reset takes time, a lot of time — several minutes — for large inventory files. But, it ran in the middle of the night, so who cares?

SQL: Worth the effort?

What does SQL offer? For standard file i/o operations (insert, update and delete), actually very little. Yes, judicious use of Prop:SQL has the potential to transfer some operations from the client PC to the server. This transfer means that network traffic will be reduced. But server utilization is proportionately increased.

In sum, this does not seem to offset the increased complexity of implementing SQL. For example, SQL queries are not always interchangeable; queries that run in one flavor of SQL may not run in another. And, while file corruptions may be less frequent with SQL, a dropped connection can leave spurious records in the database.

Then I took on a major system using MySQL. And I discovered a couple of interesting things.

First, *any* field I want to update (not in a Form) must be populated or "hot." Unless. Unless I did my update via Prop:SQL.

For example, in a Process template procedure, this code:

```
CHK:Approved = 'Yes'
CHK:CheckPrinted = Today()
Access:Check.Update()
```

fails to correctly update the Check table if either CHK:Approved or CHK:CheckPrinted is not in the Clarion view. This code:

```
check{Prop:SQL} = 'update check set approved = ' &|
  '"Yes", checkprinted = "' & Format(Today(),@d12) &|
  '" where check_sid = ' & CHK:Check_Sid & ';'
```

always succeeds. Score one for SQL: using the engine means I don't have to remember to set a field as "hot" in a Process or Report procedure.

Also, as with the inventory reset above, using the SQL engine returns control to my program more quickly. Sometimes this gain in responsiveness is even noticeable.

Second, totaling is virtually instantaneous. Where browse totaling, using the built in totaling of the Browse template, might take 15, 20 or 30 seconds, Prop:SQL could get me the total in nothing flat. Now, *this* impressed me. Two points for SQL.

Instead of looping through an entire file and adding up my source field, I can issue this command:

```
select sum(AMT) from CUSPLU;
```

and I am done. I have total PLU sales. The performance impact is so slight that I have no concern whatsoever about calling this query every time the browse returns from its update form (ResetFromAsk).

Third, reports, especially summary reports, look to be an absolute breeze using SQL due to the engine's built in aggregation:

```
select PLU,
  sum(AMT),
  sum(Cost),
  sum(QTY),
  Count(Distinct ReceiptNumber)
  from CUSPLU group by PLU;
```

And, again, I was done. I have total sales by PLU, total quantity sold, total number of receipts on which the item was sold ("number of times sold"). I can easily compute profit by subtracting sum(Cost) from sum(AMT). In fact, I could put this computation right in my Select statement:

```
select PLU,
  sum(AMT),
  sum(Cost),
  sum(AMT - Cost) as profit,
  sum(QTY),
  Count(Distinct ReceiptNumber)
  from CUSPLU group by PLU;
```

If I want to limit by SaleDate, I just add a Where clause with the appropriate date filter. If I want to include the item description from Inventory, I specify a join using (PLU) (in MS SQL: join on inventory.plu = cusplu.plu) and include the description field in my select statement.

Adding an "order by" clause, I can get my data sorted by total sales (ascending or descending), profit, quantity or any other field in the table or in any table referenced in the select statement. I can even construct the select statement under program control to allow the end user to choose the order at run time.

Summary reports, like this, are not difficult using standard Clarion reports. But, they do require a certain degree of work. Basically, such reports require either multiple passes through the source file to loop and do the subtotaling and then print or tricking the report engine into not printing a detail line for each record of a sale (another frequent topic on the news groups and the subject of several Clarion Magazine articles).

Two pass reports, looping through a file or file plus related files to create a temp file, and then looping through the temp file to print a report are no longer things to be avoided. I have finally found something that *really* makes the complexity worthwhile Or, so it seemed.

The problem

The problem is that I can write my SQL queries (just a fancy way of saying "script") in the MySQL Administrator, MS SQL Server's SQL Server Management Studio or whatever UI a vendor may provide (there are even third party UIs for this purpose). I can test/debug them. I can verify my results.

I find this extremely useful. I can be confident of my SQL without having to run the code/compile/run cycle constantly.

I can copy and paste them into an embed, preparatory to assigning them to a <*table*>{Prop:SQL}.

The one thing I cannot (that list of all the things I "can" do just had to end, didn't it?) do is use the information in my Clarion application. I cannot put CUSPLU{Prop:SQL} = in front of these SQL statements and make my Clarion report work (actually, the report did work, in that it did not GPF or exhibit any other bad behavior; but it terminated with "No Records to Process"). I cannot assign select sum(AMT) from CUSPLU to a local variable to display the total on my browse window. I cannot even get

```
myLong ='select sum(AMT) from CUSPLU;'
```

or

```
myLong{Prop:SQL} ='select sum(AMT) from CUSPLU;'
```

to compile.

Note that there are two very different kinds of things I've talked about here. First is what looks like a simple assignment, the total sales case. Second is getting an entire group (set) of records for a report. SQL may be "set oriented," returning a set of data, but when I go to use SQL in one case I expect to get a datum (singular) and in the other I expect data (plural).

This frustration, the ability to issue SQL statements but not have the results in hand, is exactly where Clarion developers were when the ODBC driver was introduced in Clarion for Windows. This dilemma was first unraveled by Troy Sorzano. He devised what he called "the dummy table technique," long since adopted by many if not most Clarion developers.

Summary

I had finally found a really motivating use for SQL. But, as soon as I fell in love with SQL, I found I could not capitalize on it. I could not get data from SQL queries into variables under my control.

Next time I'll look at how this problem can be solved.

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.

Reader Comments

Posted on Friday, February 15, 2008 by Stephen Ryan

i alway appreciate articles and comments by Mr Parker.

many thanks and looking forward to the next article.

Posted on Friday, February 15, 2008 by Michael Gorman

Steve:

Good to see you writing about SQL. You've hit upon the key issue. How can you get a "record at a time" paradigm to operate against a "set at a time" paradigm.

I would like to suggest that one possible solution is to create temporary tables that are the result set from a large SQL-engine driven database, and then have Clarion's 3GL record-at-a-time process operate against that.

The advantages for sure are that the only records in the temporary table would be those that met the select statement. Thus, there wouldn't be a series of select statements that would produce one record at a time per select statement.

Hope to read more about how to solve this paradigm conflict that exists in virtually all programming languages that work with SQL-engine databases.

Regards,
Mike G

Posted on Friday, February 15, 2008 by Steven Parker

Miguelito.

Patience (it's a virtue, I hear) <seg>.

Besides, the remaining two installments are already written, submitted, edited, folded, spindled and just waiting to go.

.....

Posted on Saturday, February 16, 2008 by Djordje Radovanovic

Very, very basic approach. I expect some more of SQL and Clarion but I think it would not impreesed me anyway.

Add a comment

Clarion Magazine

Ergonomics For Programmers

by Robert Johnson

Published 2008-02-12

All too often programmers ignore ergonomics when setting up an office. They'll spend time and energy researching what is the best computer system, which monitor to choose and which software will do the job, but often don't give much, if any, thought to the physical stresses they'll experience if their desk, chair, monitor, keyboard, or mouse are incorrectly positioned.

Of course, even if you have the perfect ergonomic office there are no guarantees that you won't succumb to Repetitive Stress Injury (RSI) or, more controversially, the dreaded Carpal Tunnel Syndrome (CTS) anyway. But hopefully this article will help you give a little more attention to workplace ergonomics and thereby help prevent RSI and/or CTS.

Definition

The term *ergonomics* is derived from two Greek words: *ergon*, meaning work, and *nomoi*, meaning natural laws. Wikipedia defines ergonomics as "the applied science of equipment design intended to maximize productivity by reducing operator fatigue and discomfort. The field is also called biotechnology, human engineering, and human factors engineering."

History

As the Wikipedia article indicates, the ancient Greeks seem to be the father of the science of ergonomics. Evidence indicates that in the 5th century BC the Hellenic civilization used ergonomic principles in tool design, jobs and workplaces. More recently, in the 19th century a Frederick Winslow Taylor discovered you could triple the amount of coal a man shoveled by *reducing* the size and weight of the coal shovel. These same principles were applied during WWII to airplane cockpit design to reduce confusion and make controls logical and usable. The list goes on and on.

Workstation ergonomics

In the introduction I mentioned Repetitive Strain Injury (RSI) and Carpal Tunnel Syndrome (CPS) as two dangers for computer programmers. Wikipedia defines RSI this way:

> A repetitive strain injury (RSI), also called cumulative trauma disorder (CTD), occupational overuse syndrome**,** or work related upper limb disorder (WRULD), is any of a loose group of conditions resulting from overuse of a tool, eg. computer, guitar, knife, etc. or other activity that requires repeated movements. It is a syndrome that affects muscles, tendons the hands, arms and upper back. The medically accepted condition in which it occurs is when muscles in these areas are kept tense for very long periods of time, due to poor posture and/or repetitive motions.

CTS is a specific kind of RSI affecting the wrist. Here again is Wikipedia:

> Carpal tunnel syndrome (CTS) or Median Neuropathy at the Wrist is a medical condition in which the median nerve is compressed at the wrist, leading to pain, parenthesia, and weakness in the forearm and hand. A form of compressive neuropathy, CTS is more common in women than it is in men, and, though it can occur at any age, has a peak incidence around age 42. The lifetime risk for CTS is around 10% of the adult population.

The authors of the Wikipedia article go on to note that CTS is often blamed on the use of computer keyboards, but there doesn't appear to be solid evidence that this particular kind of repetitive motion causes CTS.

Whether CTS itself is a likelihood for computer programmers, there's no doubt that repetitive motions and strains can cause debilitating problems. If you have any ongoing pain which could be related to your office ergonomics, you should see your primary care physician.

The purpose of this article, however, is to provide some guidance on prevent any kind of RSI in the first place, so please read on.

Set up your environment

I'll talk about some specific ergonomic recommendations in a moment, but first let me draw you a picture of my office. I am very fortunate to be self employed, and fully in control of my office environment. When our last child moved out of the house, I commandeered his bedroom for my office. I have an old beat-up 1930's era wood desk that I have modified so the keyboard and mouse are at the proper height.

I prize this room in part because of the large window which provides natural light and view of the neighborhood. I placed my 22" LCD monitor on the desk in such a way that there is no glare from outside light. My tower is just to the right of the monitor within reach when I need to insert a CD-ROM disk or plug something into the USB port. I placed the printer away from me for a reason: so I would have to get up to retrieve a printout. That forces me to get out of the chair and stretch. It also minimizes the exposure to toner from the laser printer.

My telephone is equipped with a speakerphone so I'm not tempted to cradle the handset between my neck and shoulder while using the computer. I also have a headset for when I use Skype. I have invested in a good quality chair with arm rests and have made the proper settings. I chose the overhead lighting to minimize glare and heat. All required papers, books are within arm's reach and I have plenty of desk space available to work. (You know how messy we programmers can be with printouts, manuals strewn all over the place. At least I am. The only time my office looks decent is between projects.)

I also have a long table on my left with additional space for the server monitor, TV/DVD (For training purposes only of course!), network equipment, space for my laptop when necessary, and room for more papers. All of these items are within arm's reach. I even painted the walls a somewhat soothing blue; the ceiling is white. I have my favorite paintings and movie posters on the wall in front of me when I need to "stare off into space to think".

The point is that everything is set up with ergonomics in mind, not just esthetics. An office that looks good is secondary to me; my health has first place.

If you are employed by someone your choices may be limited to what the company has to offer. However most enlightened companies recognize the value of ergonomics, and will spend money, within reason, to set up your work environment for your comfort.

For example, Paychex Inc is a company that has taken a strong interest in ergonomics. Each branch office has an appointed safety officer available to assist employees with the proper setup of their workstation. This includes setting the correct chair height for the desk; a foot platform is available so short people don't have to dangle their feet. The chairs have arm rests so the user's arms can lie in a straight line from the elbow to the keyboard. The chairs are of good quality, providing adjustable lumbar support. Wrist pads for support are provided. Sun and overhead lighting glare are considered and taken care of with shields or other means. Headsets for telephone use put a stop to the cradling-the-handset-in-the-neck problem. In a nutshell, the company strives to custom fit the workstation to each employee. Regular ongoing training about the importance of ergonomics is mandatory for all employees. Paychex knows that an "an ounce of prevention is worth a pound of cure".

Guidelines

Here are some specific guidelines to make your office safer and more comfortable.

Monitor

- To avoid eyestrain, position your monitor within a comfortable viewing distance of 18"-25" , at or below eye level, and within your eyes' 60 degree viewing field.
- If you are referring to paper documents while you work, position those documents in the line with your screen, and no more than 35 degrees to either side, to avoid excessive turning or bending of your neck.
- Place input devices (mouse, trackball) to avoid strain on your wrists, arms and hands.
- Try to pick a spot away from sources of glare such as windows, overhead lights, and shiny surfaces.
- Place your monitor so that the screen is perpendicular to any windows; this will help avoid reflections (it may be necessary to draw the shades or adjust the blinds at times).

Work Surface

- Standard height work surfaces, at 29"-30", will be too high for intensive keyboard or mouse usage. Your keyboard and mouse height should be in the 26"-28" range. If you have a desk or work surface that will allow installation of a keyboard shelf look for one that will allow room for your mouse also.
- If you must place your keyboard and mouse on the desk top, try raising your chair to provide a more comfortable arm angle (an approximate 90 degree angle between upper and lower arm is recommended) or use a keyboard tray. Raising the chair may also require a foot rest.
- Position task lights as necessary to prevent glare on the monitor.
- The keyboard should be located at approximately seated elbow level with hands an wrist in a straight or neutral position during use.
- Consider a split keyboard, which allows a more natural position of the wrists. While they do take a little getting used to, converts say they won't go back.
- I recommend trackball style mouse over a traditional mouse for one simple reason: less hand movement. Of course it is also a matter of taste, but some Clarion developers have reported significant relief from pain after switching to a trackball. The Kensington Expert Mouse appears to be a popular choice.

Chair

- Look for a chair that is stable and easily adjustable from a seated position.
- Choose one that provides good support for your body, especially the lower back.
- If your chair is not one you can replace (the company supplied it) and it's not comfortable, try placing a small cushion or a rolled up towel behind your lower back for additional support.
- Adjust the height of your chair so that your thighs are horizontal, your feet rest flat on the floor, and your arms and hands are comfortably positioned at the keyboard. Use a footrest, box, or large book to support you feet of they do not rest flat on the floor when your chair is properly adjusted.

Working comfortably

Modify the placement of your chair, keyboard, and monitor to find what suits you best. There is no "best" arrangement; find the one that works for you. Try some of the suggestions below to minimize fatigue and discomfort in your workplace.

- Change your seated position occasionally and stand up and stretch frequently to ensure good circulation.
- Use a soft touch on the keyboard and try to keep your shoulders, hands, and fingers relaxed.
- Use a document holder to position source documents at the same height and distance as the screen.
- Clean your screen and eyeglasses (if you wear them) regularly. LCDs need special care, as pressure on the screen can damage it. Follow the manufacturer's instructions.

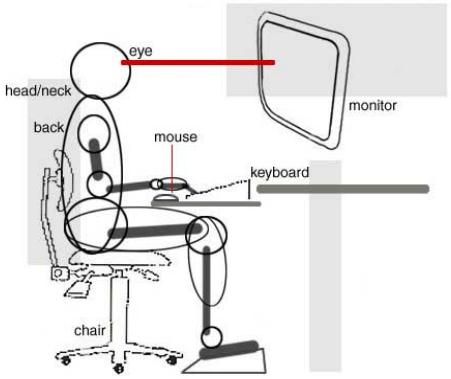

Figure 2. Workstation setup (image courtesy of the University of Texas Libraries, The University of Texas at Austin)

Prevention

Take frequent breaks from repetitive keyboard use; get up and stretch your spine and wrists. There are free software programs like Workrave (Windows/Linux) and Xwrits (Linux) that will remind you to take your breaks.

Typing technique - Your arms should "float" over your keyboard - your wrists/palms should not be resting on the desktop or even on a wrist rest (unless you are breaking between typing spurts). Keep your wrists straight and hands relaxed (this is true when using your pointing device, as well). Hit the keys lightly.

Here are some things that may help prevent RSI - CTS:

- Lose weight if you're overweight.
- Get treatment for any disease you have that may cause CTS.
- If you do the same tasks with your hands over and over, try not to bend, extend, or twist your hands for long periods.
- Don't work with your arms too close to or too far from your body.
- Don't rest your wrists on hard surfaces for long periods.
- Switch hands during work tasks.
- Take regular breaks from repeated hand movements to give your hands and wrists time to rest.
- Don't sit in the same position all day.

Eye strain tip (courtesy of IBM Corp)

Focus: Preventing eyestrain

While standing or sitting:

1. Look out a window or as far away from work area as possible
2. Focus on a far-away object
3. Move your eyes around to look at other objects
4. Look back at computer screen
5. Repeat often throughout workday

Conclusion

When I first set up in business in 1987 I had heard about ergonomics from my previous employer. I did research and set my home office with ergonomics in mind. I'm convinced that in all my years at the keyboard I have never once

experiencing RSI or CTS because of my interest in ergonomics from the beginning. I wasn't fanatical about it, and on occasion broke the rules, but for the most part I tried to follow the principles behind an ergonomic office.

Those of us who have done consulting have probably been in offices where we were appalled by the ergonomics of employees' workstations. On several occasions I have spoken to owners/supervisors about the subject; sometimes it falls on deaf ears and sometimes it bears fruit. All I can say is I have no pain and I truly believe it's because I paid attention a long time ago. And remember, it's never too late to start.

References

- Wikipedia.org
- IBM
- University of Texas
- Workrave
- xwrits

Robert Johnson began programming in 1987, writing Lotus 123 spreadsheets and selling business modeling spreadsheets to his first client, Entrepreneur Magazine Inc. He purchased Clarion DOS Version 2.03 (If he remembers the version correctly) on a friend’s recommendation and hasn’t looked back since. He has worked in VB1, VB3, C#, VB.NET, PHP, and Drupal, but Clarion is his primary language of choice.

Reader Comments

Posted on Tuesday, February 19, 2008 by Martin Howes

Good article.

Peace and quiet is important. Other people's conversations can be a significant distraction. Other people's choice of music can also be annoying!

Good sound insulation is also needed when discussing confidential matters with clients.

And for the times when you really need to concentrate, switching the phones to voicemail is very useful.

Add a comment

Clarion Magazine

The ClarionMag Blog

Get automatic notification of new items! RSS feeds are available for:

XML All blog entries
XML All new items, including blogs

Blog Categories

- »All Blog Entries
- »Clarion 7 Clarion.NET
- »Future Articles
- »News flashes
- »Nifty Stuff

Still to come: a batch compiler and some cool string stuff

Direct link

Posted Wednesday, February 27, 2008 by Dave Harms

There are two articles in final review, and I expect them to show up on Friday. One is a nifty drag and drop batch compiler by Richard Rose, which demonstrates both D&D and DDE (the means by which you tell the Clarion IDE to load and compile apps). The other is another look at ClaString and String by yours truly, now that the dust has settled on the Clarion# class instantiation and array index changes.

A new Clarion.NET build

Direct link

Posted Monday, February 18, 2008 by Dave Harms

Build 2957 has gone out to beta participants. This is a pretty big release, since it contains some significant language changes as well as the first fully functional dictionary editor. Specific features include:

- Fully enabled dictionary editor - insert, delete, change, update all items
- No more auto-instantiation, and for the most part no more need to use & in declarations

- Zero based indexes
- := operator no longer used; instead, use the = operator
- Deep assignment operator added
- Optional parenthesis for NAMESPACE and USING
- USING alias support
- Generic classes can be used (but not yet declared)
- NEW can be used as a modifier to hide an inherited member

There are a ton of bug fixes and other changes. I'll have a closer look at some of the new stuff in an upcoming article.

The other kind of bug

Direct link

Posted Friday, February 08, 2008 by Dave Harms

I'm a little slow off the mark this month with articles, PDFs, source zips, and the update to the source code library. Until this week I was taking some pride in having avoided the cold and flu bugs that have hit my family, but we all know what pride cometh before. As a result I've been out of action much of this week. But the PDF and source zip should be up shortly, followed by the source code library update. And there's a good lineup of articles for this month as well, although the first of those won't likely appear until the middle of next week.
