Clarion Magazine

Clarion News

- » Huenuleufu Name Change
- » Clarion.NET Build Released
- » RPM for C7.1
- » vuFileTools 3.5 Beta
- » C7.x Third Party Deployment Script
- » C7.1 Installation Test Tool
- » CPCS 7.10 Installers
- » KSpng 7.1 Compatible
- » SetupBuilder 7.1 Pre-Release
- » CalendarPro Wrapper Template 2.02
- » Locus Templates C7.1 Compatible
- » CPCS 7.10
- » Fomin Report Builder 3.12
- » EasyCOM2INC 2.11
- » RPM/AFE Introductory Subscription Prices End Dec 31
- » Updated RPM 7 and 6.3 Installs
- » iQ-Sync 1.13
- » PropertyGrid Wrapper 1.12
- » KSpng Class Updated
- » DMC Subscriptions
- » Clarion 7.1 Release Update
- » DockingPane Wrapper 1.07
- » Twitter Remote Control With C6/NetTalk
- » EasyNaviBar 1.00
- » Ingasoftplus Time Limited 25.00% Discount
- » CHT Web Client Server Videos
- » CHT Build 13D1.00

[More news]

- » Clarion.NET FAQ

Latest Subscriber Content

ClarionMag on ClarionLive! Everything you ever wanted to know about Clarion Magazine and more...

This Friday (Dec 4) let Dave Harms, Clarion Magazine's editor and publisher, take you on a tour of the web's #1 Clarion resource. Learn how to use the search engine effectively, locate articles via the topical index, get a single page listing of 1379 ClarionMag articles, browse the Clarion Online archives, view free articles, get up to the minute notices of changes with RSS feeds, and more. You're sure to discover something you didn't know about ClarionMag! Dave is also hard at work on a massive rewrite of the Clarion Magazine site, so bring your suggestions for what you'd like to see in the next version of ClarionMag. The webinar starts at 9 a.m. Pacific time.

Posted Wednesday, December 02, 2009

Source Code Library 2009.11.30 Available

The Clarion Magazine Source Code Library has been updated to include the latest source. Source code subscribers can download the November 2009 update from the My ClarionMag page. If you're on Vista or Windows 7 please run Lindersoft's Clarion detection patch first.

Posted Monday, December 07, 2009

ClarionMag on ClarionLive! Embeds, OOP, TDD and more! Dec 18!

Join Dave Harms Friday, December 18 (NOTE: rescheduled from Dec 11) at ClarionLive! (9 a.m. Pacific time) and wrap your mind around OOP, Unit Testing, and Test Driven Development. Yes, you really do need to learn this stuff. It's time, it's not as hard as you think, and the benefits are huge. This webinar is a companion to the current series of articles in ClarionMag titled "The Problem With Embeds." Watch Dave fearlessly rip some truly gnarly embed code out of the Invoice example app and replace it with an easy-to-write, highly testable class.

Posted Wednesday, December 09, 2009

The Problem With Embeds, Part 4: The CalcValues Code

Dave Harms continues his series on the problem with embed code by rewriting the CalcValues routine from the Invoice app as a class.

Posted Monday, December 14, 2009

The Problem With Embeds, Part 5: Unit Testing The InvoiceDetail Class

In this fifth article in his series on the problem with embed code, Dave Harms writes a number of unit tests to verify that the InvoiceDetail class is calculating values correctly.

Posted Monday, December 14, 2009

Converting Your Apps To C7

Russ Eggen covers some of the more common errors you can encounter when converting applications from C6 to C7, including window structure errors and phantom templates.

Posted Monday, December 21, 2009

SQL Tips and Tricks

Joe Tailleur shares some of his SQL tips and techniques, including field naming, dummy tables, speeding up reports, self-

- » Clarion# Language Comparison

[More Clarion & .NET]

[More Clarion 101]

Latest Free Content

- » ClarionMag on ClarionLive! Everything you ever wanted to know about Clarion Magazine and more...
- » ClarionMag on ClarionLive! Embeds, OOP, TDD and more! Dec 18!
- » 75 Clarion Questions And Growing!

[More free articles]

Clarion Sites

Clarion Blogs

joins, and more.

Posted Wednesday, December 23, 2009

First Look: Clarion 7.1

SoftVelocity has released Clarion 7.1, which includes a long list of bug fixes and new features, such as a full-fledged window previewer and beta release of the new report writer. Is this the C7 release we've all been waiting for?

Posted Thursday, December 31, 2009

Clarion: A Decade In Review

It's a wrap for the first decade of the new millenium. Clarion Magazine's editor takes a look back at the last ten years in ClarionLand.

Posted Thursday, December 31, 2009

75 Clarion Questions And Growing!

Sponsored by Clarion Magazine, the free-access ClarionFAQ site lets you ask and answer all sorts of Clarion-related questions. ClarionFAQ is already at 75 questions (many with multiple answers) and picking up steam.

Posted Thursday, December 31, 2009

[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 Tips & Techniques Volume 5 - ISBN 978-0-9784034-1-6
- » Clarion Tips & Techniques Volume 4 - ISBN 978-0-9784034-0-9
- » Clarion Tips & Techniques Volume 3 - ISBN: 0-9689553-9-8

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

The Problem With Embeds, Part 4: The CalcValues Code

by Dave Harms

Published 2009-12-14

This article is the fourth in a series on what Clarion developers (yours truly included) do wrong with embeds, and how to do embeds right. In a nutshell, Clarion's application architecture encourages an all-in-one approach, where the user interface, the business logic, and the data access code are all lumped together.

This is a problem.

Actually it's not a huge problem for the generated code, inasmuch as that code tends to actually work as expected. It's tried and true, and if the architecture leaves something to be desired, well, so what?

It's when we start throwing our own code into embed points that things tend to get really ugly.

In Part 1 of this series I examined the architecture of Clarion applications; in Part 2 I talked in general terms about extracting code from embed points and making it testable. In Part 3 I tackled the Invoice example application, the one you can find in your Clarion examples directory (I used the C6 version).

In this article I'll take some of the Invoice app's buggy embed code, I'll show how to recreate it as a class, I'll run a series of unit tests on that class, and I'll plug the new code back into the embed point. Most importantly, the embed point will no longer contain any business logic; it'll just have calls to the class, and the class will contain the business logic.

> **Note:** I'm scheduled to do a ClarionLive Webinar on Friday, December 18, 2009 covering many of the points discussed in this article. Please join me, and if you miss the webinar you can always download the recording.

The Invoice app

The specific code I'm interested in is in the UpdateDetail procedure in the Invoice app. It's a routine that's used to calculate various values for each line item that's added to an invoice. As I'm sure you'll agree, this is critical business logic. And as I showed in Part 3, it's buggy code.

Let's take a look at what this embed code is actually trying to accomplish. Here again is the code:

```
CalcValues  ROUTINE
  IF DTL:TaxRate = 0 THEN
    IF DTL:DiscountRate = 0 THEN


      DTL:TotalCost = DTL:Price * DTL:QuantityOrdered
    ELSE
      LOC:RegTotalPrice = DTL:Price * DTL:QuantityOrdered
      DTL:Discount = LOC:RegTotalPrice * DTL:DiscountRate / 100
      DTL:TotalCost = LOC:RegTotalPrice - DTL:Discount
      DTL:Savings = LOC:RegTotalPrice - DTL:TotalCost
    END
  ELSE
    IF DTL:DiscountRate = 0 THEN
      LOC:RegTotalPrice = DTL:Price * DTL:QuantityOrdered
      DTL:TaxPaid = LOC:RegTotalPrice * DTL:TaxRate / 100
      DTL:TotalCost = LOC:RegTotalPrice + DTL:TaxPaid
    ELSE
      LOC:RegTotalPrice = DTL:Price * DTL:QuantityOrdered
      DTL:Discount = LOC:RegTotalPrice *  DTL:DiscountRate / 100
      LOC:DiscTotalPrice = LOC:RegTotalPrice - DTL:Discount
      DTL:TaxPaid = LOC:DiscTotalPrice * DTL:TaxRate / 100
```

```
        DTL:TotalCost = LOC:DiscTotalPrice + DTL:TaxPaid
        DTL:Savings = LOC:RegTotalPrice - DTL:TotalCost
      END
    END
```

First, there are a number of different states, which can be expressed as a matrix:

	Not Discounted	**Discounted**
Not Taxed	price * qty	price * qty - discount
Taxed	price * qty + tax	price * qty - discount + tax

Any item can have a discount, and it can have a tax, and it can have both a discount and a tax. If a taxed item is discount, we'll assume that the tax is applied *after* the discount (although in real life this isn't always the case, as with manufacturer's rebates in which the tax may apply to the non-discounted total - but beyond the scope of this article).

One of the problems with the embedded code is terminology. It isn't especially clear on the concept of an *extended price*, which is the unit price (price for one item) times the quantity. Instead, there's a local variable called LOC:RegTotalPrice, which is calculated in three of the four matrix elements. For the simplest case (no taxes, no discount) it isn't used at all; instead the calculation just sets the total price to the price times the quantity.

There are four values that care calculated:

- Total Cost - the actual amount paid by the customer
- Discount - the dollar value of the discount
- Tax Paid - the dollar value of the tax
- Savings - this one is a little unclear, but it appears to be the extended price minus the total cost (yielding a negative number, with a lower number meaning greater savings). That doesn't make a whole lot of sense to me when taxes are involved, since the higher the tax the greater the perceived savings! I can sooner see the savings as including any tax savings realized by the application of the discount (and so being the same as the discount amount if the tax rate is zero), but

as written the code isn't going to produce that value.

So to start with let's use a formal definition of an extended price:

extended = price * qty

Now, how about discount amount? That's simply the extended price times the discount rate:

discount = extended * discountrate

Similarly, the tax amount (assuming no discount) is the extended price times the tax rate:

tax = extended * taxrate

And if there is a tax and a discount, the tax is the discounted amount times the tax rate:

tax = (extended - discount) * taxrate

And the grand total is simply the extended amount, less any discount, plus the tax:

total = extended - discount + tax

Here's the matrix with the updated pseudocode:

	Not Discounted	**Discounted**

Not Taxed	extended = price * qty
	
	total = extended
		extended = price * qty
		
		discount = extended * discountrate
		
		total = extended - discount

Taxed	extended = price * qty
	tax= extended * taxrate
	total = extended + tax
		extended = price * qty
		discount = extended * discountrate
		tax = (extended - discount) * taxrate
		total = extended - discount + tax

So far, so good. But how do you translate this logic into a class that can be tested and reused?

Class design and test driven development

Back when I first started writing object-oriented code, I often found the most difficult part of the job was imagining what the class would look like. Which methods should the class have? What properties? Would I need one class, or multiple classes?

And at some point I realized that it was a whole lot easier if I approached class design, not from the perspective of what I wanted to include in the class (the methods, properties), but *how I wanted to use that class*. So I'd imagine some code that called the class. And that often made the class design task much, much easier.

And then I came across Test Driven Development, or TDD. And it all looked very familiar to me, because in a way that's what I'd been trying to with my "how will I use this class" visualizations.

TDD isn't something you hear about a lot on the Win32 platform; I came across it while doing .NET development (Clarion# and C#). The basic idea is you begin writing code by writing the tests first. For instance, I might have a test that looks something like this:

```
    ! In the data section
    detail        InvoiceDetail



    ! In the code sectin

    detail.Init(12.50,3) ! 3 items at $12.50 each
```

```
extended = detail.GetExtended()
if extended <> 37.50
   ! Report an error
end
```

Here I have an instance of an InvoiceDetail class that models one invoice line item, and I'm initializing it with a price (12.50) and a quantity (3). I'm then calling a GetExtended() method on the class to get the extended price, and if that price isn't what I think it should be I'm going to report an error.

Note that at this point my code *will not compile*. That's critical, as it is due to test-*driven* aspect of TDD. I'm using the test as my *starting point*. I haven't written the class yet. So the next step will be to write a stub class so my test compiles, but of course the test will fail because I won't have any logic in the class yet. *It's only once I have a failing test that I actually start writing the class code.*

> Note: You don't have to use a test-*driven* approach to get the benefit of unit testing. You can always write your tests after you've written the code. But I think you'll find that the more you use unit tests, the more you use the tests as a starting point.

But even before I start writing any code I have to back up a bit further, because the one thing still missing from this picture is a *unit testing framework*.

A unit testing framework

A unit testing framework is a software tool that makes it easy to unit test your code. But what's a unit test?

Wikipedia defines unit testing as

> a software verification and validation method in which a programmer tests if individual units of source code are fit for use. A unit is the smallest testable part of an application.

In practice, unit testing often involves fairly complex processes, but each individual test should only test one thing. And if you start with the smallest, simplest testable units and work your way up, you can achieve a high degree of confidence that your code is working as expected. In the above example I'm testing the ability of the class to return a correct extended price; I'll go on to write tests that verify the class's ability to calculate taxes, discounts and totals.

The entry goes on to explain that unit testing is commonly automated and run in the context of a framework:

> Under the automated approach, to fully realize the effect of isolation, the unit or code body subjected to the unit test is executed within a framework outside of its natural environment, that is, outside of the product or calling context for which it was originally created. Testing in an isolated manner has the benefit of revealing unnecessary dependencies between the code being tested and other units or data spaces in the product. These dependencies can then be eliminated.
>
> Using an automation framework, the developer codes criteria into the test to verify the correctness of the unit. During execution of the test cases, the framework logs those that fail any criterion. Many frameworks will also automatically flag and report in a summary these failed test cases. Depending upon the severity of a failure, the framework may halt subsequent testing.
>
> As a consequence, unit testing is traditionally a motivator for programmers to create decoupled and cohesive code bodies. This practice promotes healthy habits in software development. Design patterns, unit testing, and refactoring often work together so that the best solution may emerge.

Unit testing under .NET is a treat - there are some terrific tools out there that make it easy to create, run, and log tests. You can run tests manually or have them run automatically as part of your build process (typically via a continuous integration server, if you take that approach).

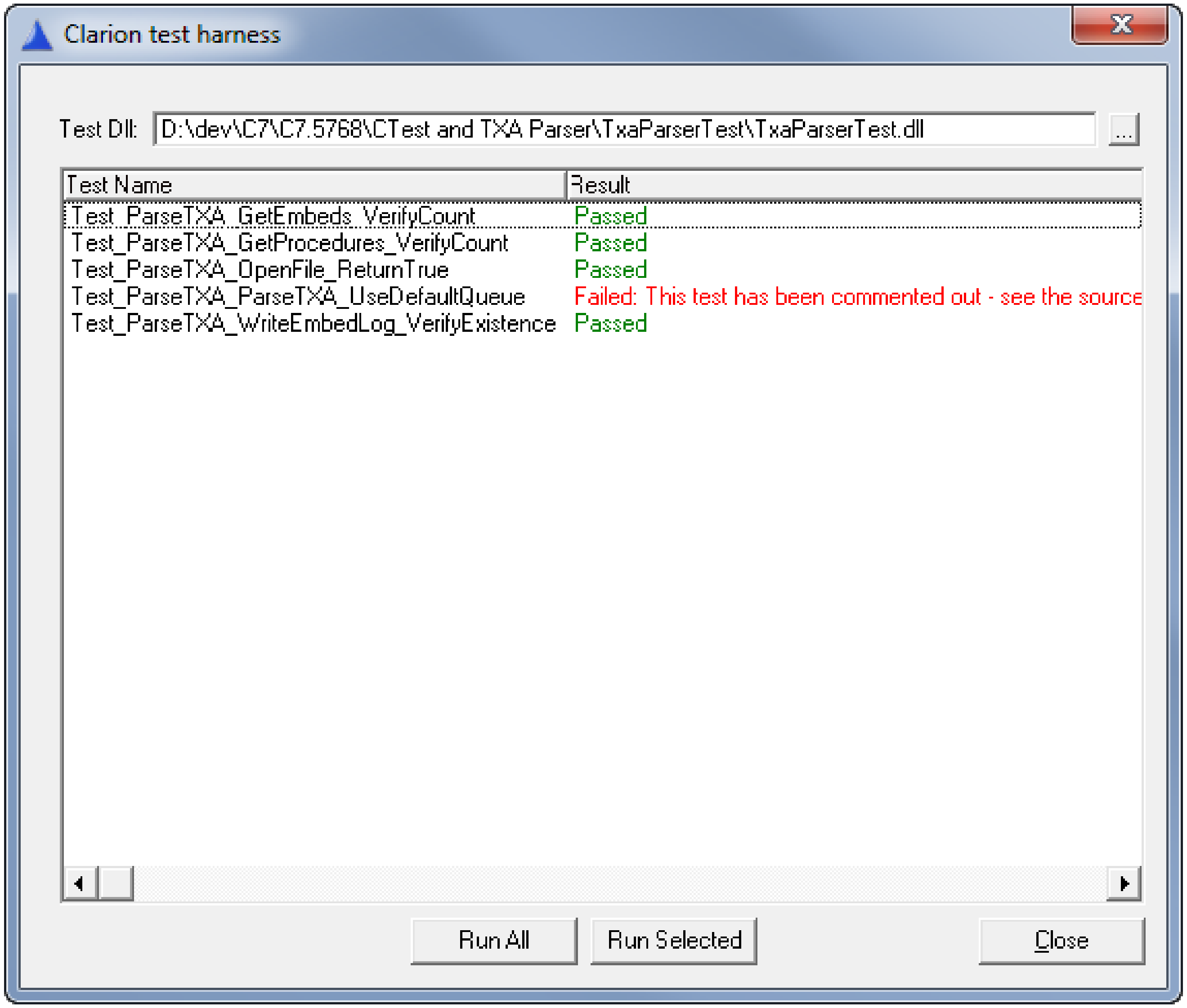
But .NET unit testing tools benefit from some .NET language features, particularly reflection. A .NET unit test framework can inspect an assembly, locate test methods inside test classess, run those test methods and report on the results. That's a lot more difficult to accomplish in Win32.

Unit testing in Clarion Win32

I wasn't aware of any third party unit testing tools for Clarion, so I went ahead and wrote one. Actually this tool is in two parts: an executable application (called CTest) that will search a DLL looking for unit tests to run, and a procedure template you use to create test procedures, in your test DLL, that are compatible with CTest.

Figure 1 shows the CTest application with a test DLL loaded and several tests having been run (this screen shot is from Part

2). As you can see, CTest is just a single window application that lists the available tests, lets you run those tests, and shows you the results.



Test Dll: D:\dev\C7\C7.5768\CTest and TXA Parser\TxaParserTest\TxaParserTest.dll

Test Name	Result
Test_ParseTXA_GetEmbeds_VerifyCount	Passed
Test_ParseTXA_GetProcedures_VerifyCount	Passed
Test_ParseTXA_OpenFile_ReturnTrue	Passed
Test_ParseTXA_ParseTXA_UseDefaultQueue	Failed: This test has been commented out - see the source
Test_ParseTXA_WriteEmbedLog_VerifyExistence	Passed

Figure 1. The CTest test runner

The Test_ prefix is necessary for CTest to identify the procedure as a test procedure (this way you can mix test and non-test procedures in the DLL).

I'll get into the specifics of how to use CTest and the test procedure template shortly (and I'll have an article on CTest later this month).

To recap, here are the stages I've completed:

- I've analyzed the embed code enough to figure out more or less what it's supposed to do
- I've come up with a rough idea for my first unit test

Now I'm about to:

- Create a test DLL
- Write my first unit test procedure
- Create a skeleton class so the unit test will compile
- Run the unit test using the CTest framework
- Fix the class so it returns the expected result

I realize that this seems like a lot of work for just one wee bit of embedded code. But a lot of this work is also a one-time effort which you can leverage when you write additional tests for other code/classes. Once you understand the basic concepts of unit testing, creating new tests will come very easily.

Having said that, even after you become proficient in creating unit tests you almost certainly will spend more time up front creating your code than you did writing embed code. It takes time to write good tests. But there's also a huge payback, in fewer bugs, more confidence and better reusability.

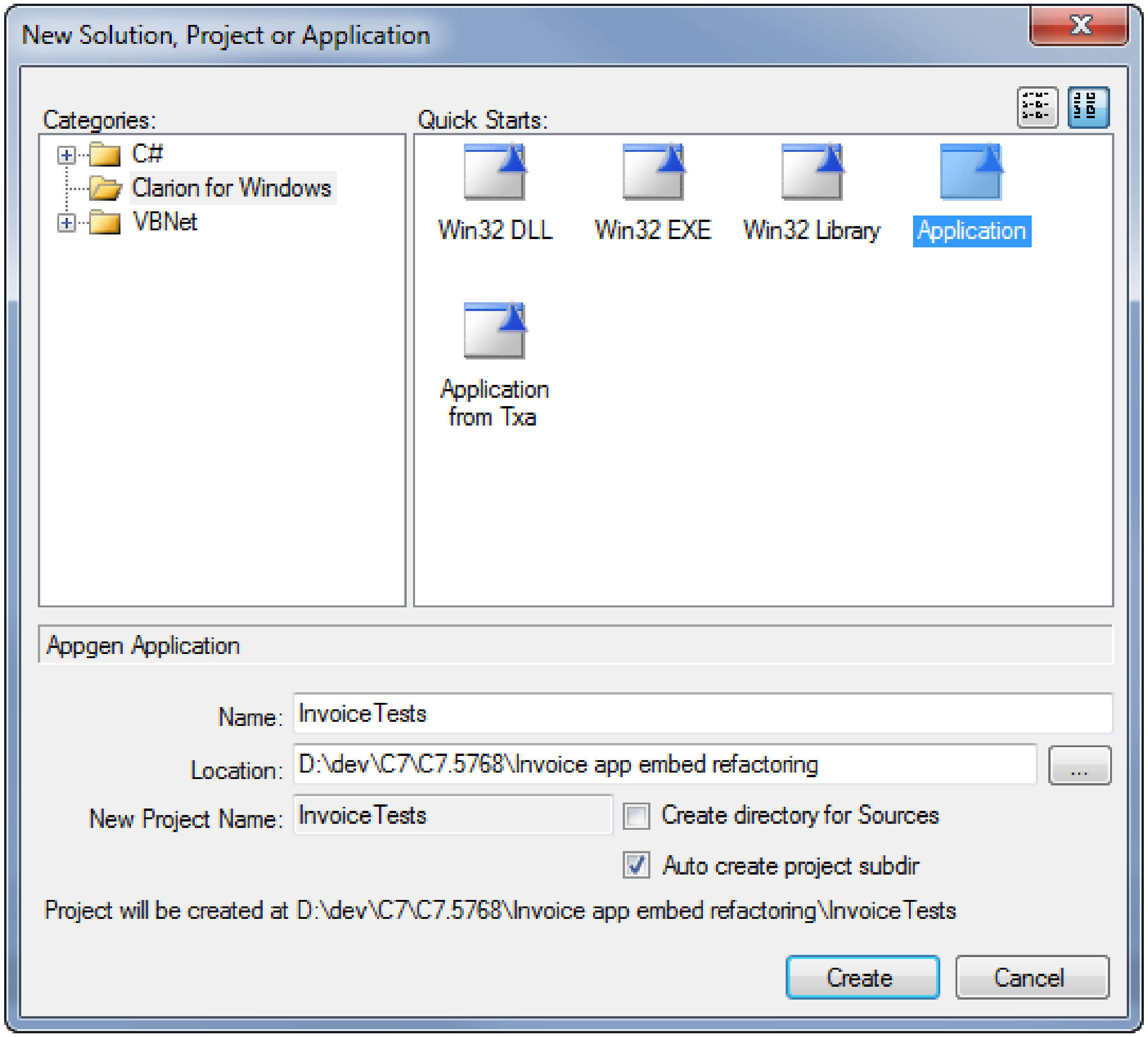
Creating a test DLL

I'm using Clarion 7 in the following explanation, mainly because it lets me create a single solution containing both the original application as well as my test DLL.

First, copy the ClarionTest.TPL file from the downloadable source into your Clarion template directory and register it. This template contains the test procedure template as well as a global extension you'll need for the test application.

The downloadable source zip contains all the projects you need (including the CTest project). Or if you're following along you can work directly with the Clarion Invoice application. I'm using the C6 version which I've converted to C7.

If you're using your own solution, right-click on the solution in the Solution Explorer and choose Add | New Application or Project. Choose a type Application (not DLL) from the dialog (Figure 2).



Appgen Application

Name: InvoiceTests

Location: D:\dev\C7\C7.5768\Invoice app embed refactoring

New Project Name: InvoiceTests

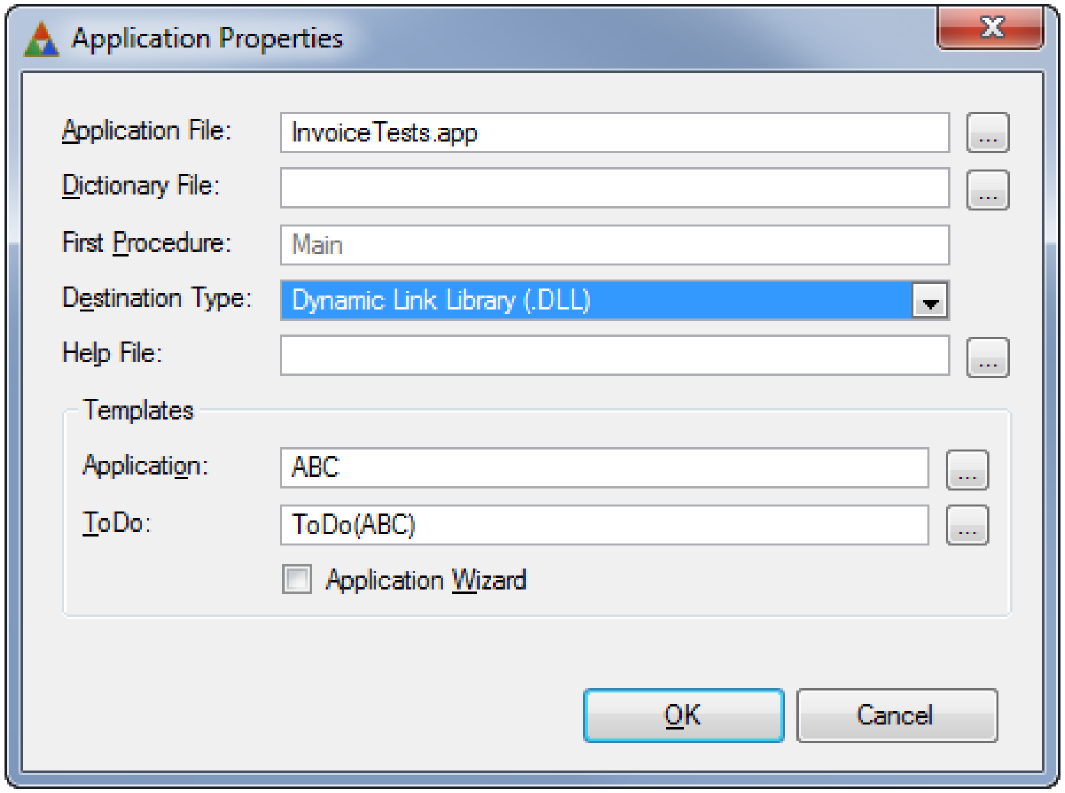
Create directory for Sources [ ]

Auto create project subdir [✓]

Project will be created at D:\dev\C7\C7.5768\Invoice app embed refactoring\InvoiceTests

Figure 2. Adding a test DLL to the solution

In the Application Properties dialog set the Destination Type to DLL (Figure 3).



Application File: InvoiceTests.app

Dictionary File:

First Procedure: Main

Destination Type: Dynamic Link Library (.DLL)

Help File:

Templates

Application: ABC

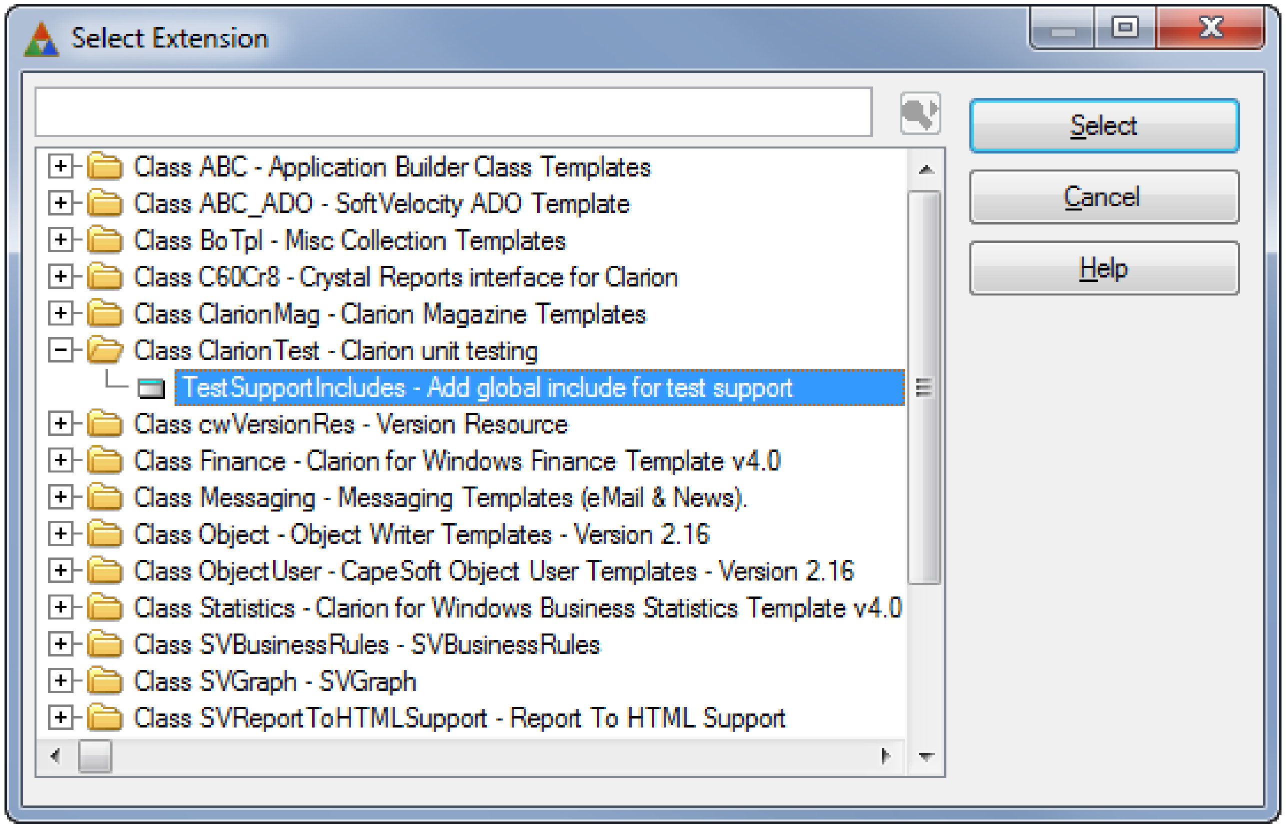
ToDo: ToDo(ABC)

☐ Application Wizard

Figure 3. Creating the test DLL

You don't need the Application Wizard since you'll be creating all your procedures manually.

You do however need to add a global extension to the DLL to enable the unit test support code. On the Global Extensions tab click on Insert, and choose Class Clarion Test | TestSupportIncludes (Figure 4).



Class ABC - Application Builder Class Templates
Class ABC_ADO - SoftVelocity ADO Template
Class BoTpl - Misc Collection Templates
Class C60Cr8 - Crystal Reports interface for Clarion
Class ClarionMag - Clarion Magazine Templates
Class ClarionTest - Clarion unit testing
TestSupportIncludes - Add global include for test support
Class cwVersionRes - Version Resource
Class Finance - Clarion for Windows Finance Template v4.0
Class Messaging - Messaging Templates (eMail & News).
Class Object - Object Writer Templates - Version 2.16
Class ObjectUser - CapeSoft Object User Templates - Version 2.16
Class Statistics - Clarion for Windows Business Statistics Template v4.0
Class SVBusinessRules - SVBusinessRules
Class SVGraph - SVGraph
Class SVReportToHTMLSupport - Report To HTML Support

Figure 4. Adding the TestSupportIncludes global extension

You're now ready to create your first test procedure, and I'll cover that topic next time.

Download the source

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

The Problem With Embeds, Part 5: Unit Testing The InvoiceDetail Class

by Dave Harms

Published 2009-12-14

In Part 4 of this series I began rewriting some of the embed code from the UpdateDetail procedure in the Invoice app as a reusable, testable class. I also showed how to create a DLL that's compatible with the CTest test runner utility. Now it's time to create a test procedure.

The first test procedure

Click on New Procedure and add a procedure called Test_CreateDetail_SetPriceAndQuantity_VerifyExtended.

In the Select Procedure Type dialog click on the Defaults tab and choose Test Procedure (Figure 5).

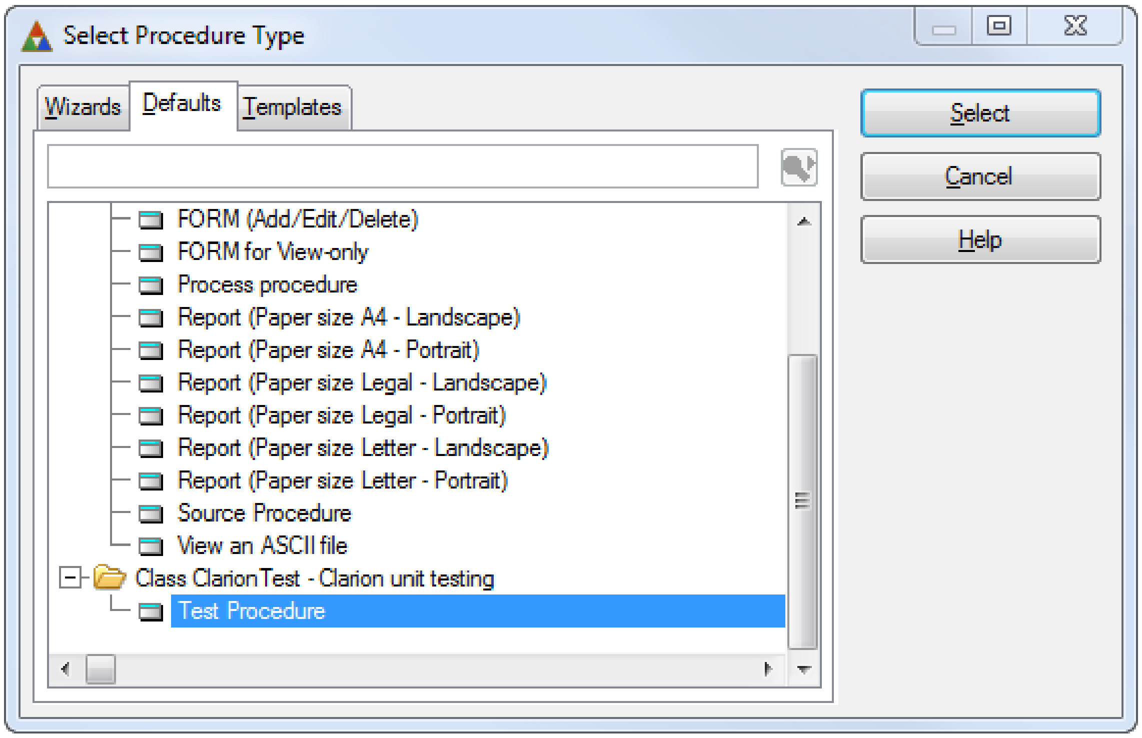
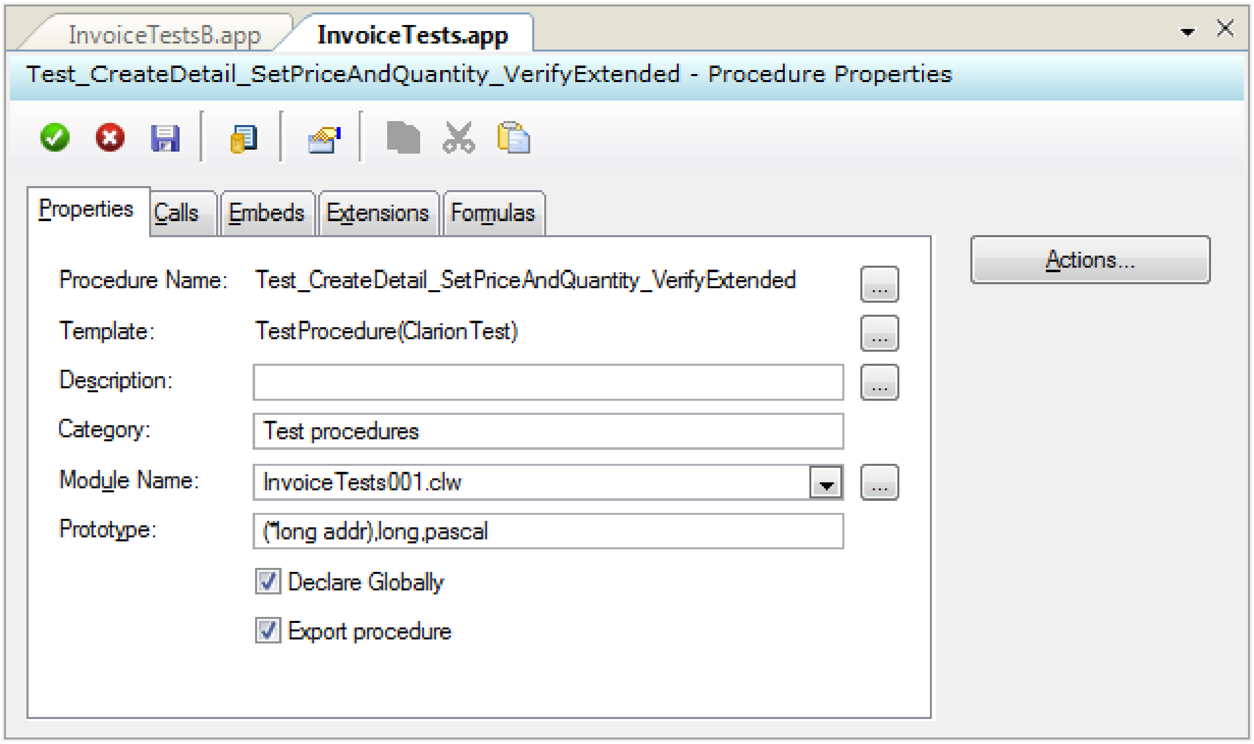

Select Procedure Type
Wizards
Defaults
Templates
Select
Cancel
Help
FORM (Add/Edit/Delete)
FORM for View-only
Process procedure
Report (Paper size A4 - Landscape)
Report (Paper size A4 - Portrait)
Report (Paper size Legal - Landscape)
Report (Paper size Legal - Portrait)
Report (Paper size Letter - Landscape)
Report (Paper size Letter - Portrait)
Source Procedure
View an ASCII file
Class ClarionTest - Clarion unit testing
Test Procedure

Figure 5. Choosing the test procedure template

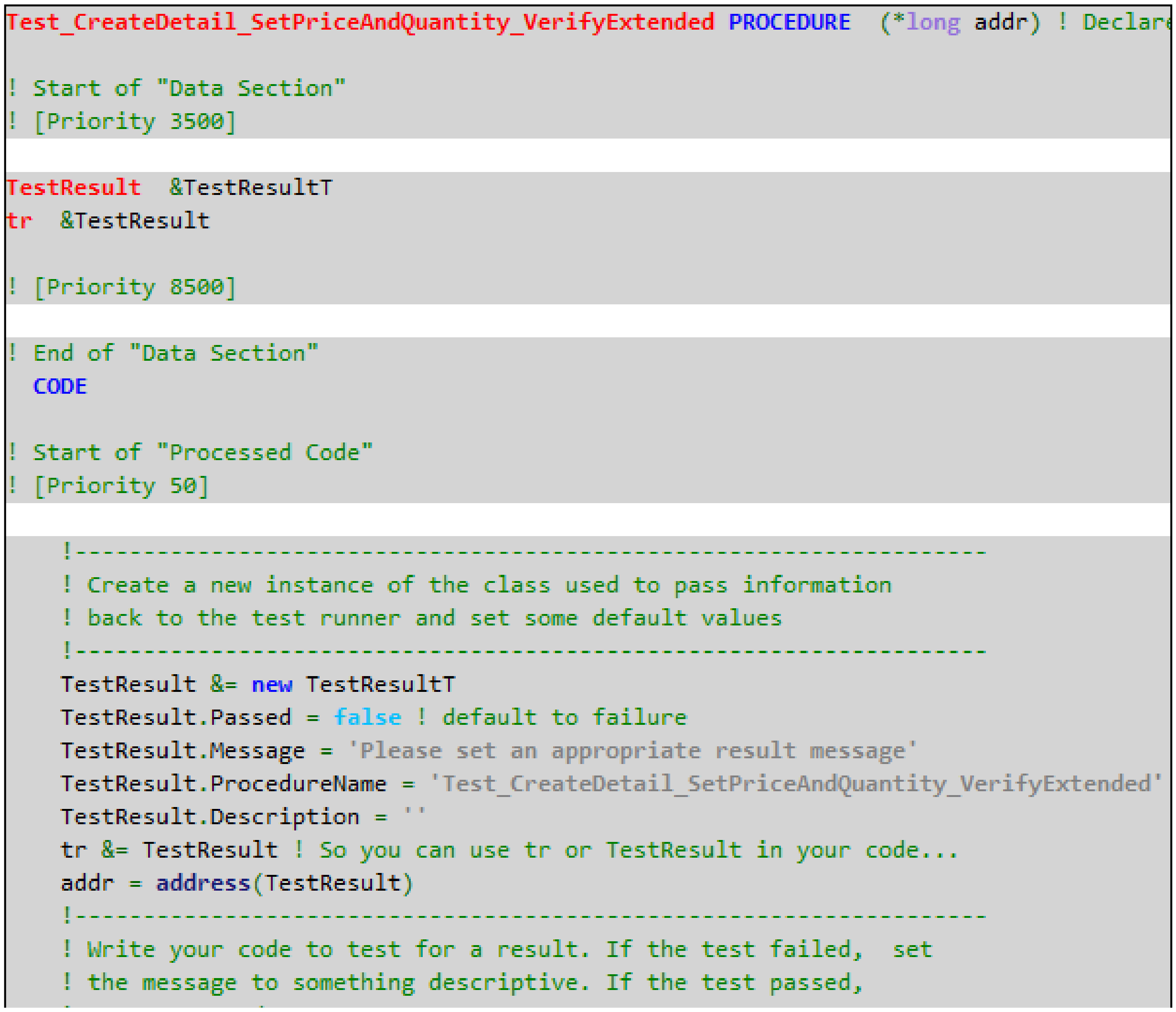
You'll notice that the procedure properties have been preloaded with a prototype and parameters (Figure 6). Do not change these values or the test procedure will not work, and may cause a GPF in CTest or your DLL.



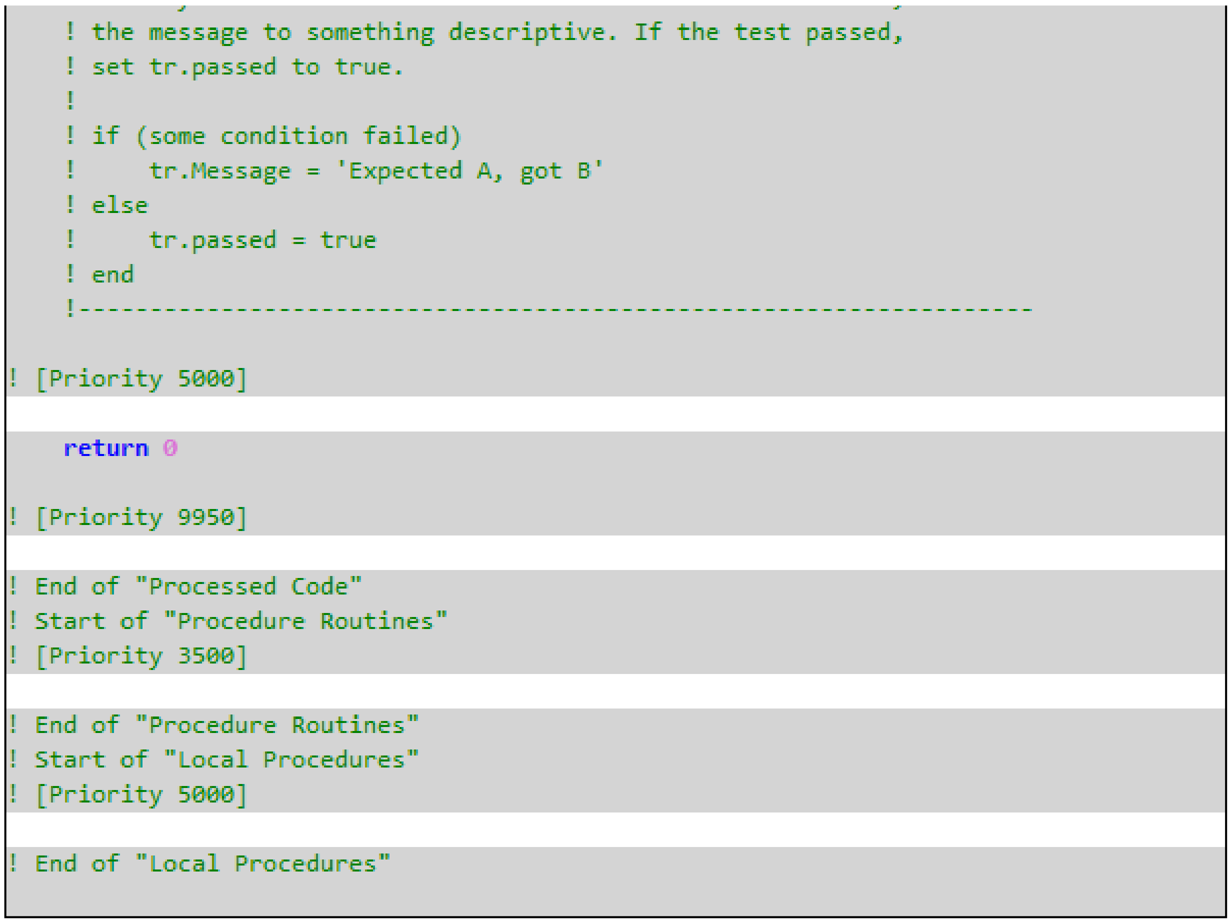
Procedure Name: Test_CreateDetail_SetPriceAndQuantity_VerifyExtended
Template: TestProcedure(ClarionTest)
Description:
Category: Test procedures
Module Name: InvoiceTests001.clw
Prototype: (*long addr),long,pascal
[✓] Declare Globally
[✓] Export procedure

Figure 6. Properties for the test procedure

Now, right-click on the procedure and choose Source to bring up the Embeditor (Figure 7).



```
Test_CreateDetail_SetPriceAndQuantity_VerifyExtended PROCEDURE  (*long addr) ! Declare

! Start of "Data Section"
! [Priority 3500]

TestResult  &TestResultT
tr  &TestResult

! [Priority 8500]

! End of "Data Section"
  CODE

! Start of "Processed Code"
! [Priority 50]

    !---------------------------------------------------------------------
    ! Create a new instance of the class used to pass information
    ! back to the test runner and set some default values
    !---------------------------------------------------------------------
    TestResult &= new TestResultT
    TestResult.Passed = false ! default to failure
    TestResult.Message = 'Please set an appropriate result message'
    TestResult.ProcedureName = 'Test_CreateDetail_SetPriceAndQuantity_VerifyExtended'
    TestResult.Description = ''
    tr &= TestResult ! So you can use tr or TestResult in your code...
    addr = address(TestResult)
    !---------------------------------------------------------------------
    ! Write your code to test for a result. If the test failed,  set
    ! the message to something descriptive. If the test passed,
```



```
    ! the message to something descriptive. If the test passed,
    ! set tr.passed to true.
    !
    ! if (some condition failed)
    !     tr.Message = 'Expected A, got B'
    ! else
    !     tr.passed = true
    ! end
    !-------------------------------------------------------------------

! [Priority 5000]

    return 0

! [Priority 9950]

! End of "Processed Code"
! Start of "Procedure Routines"
! [Priority 3500]

! End of "Procedure Routines"
! Start of "Local Procedures"
! [Priority 5000]

! End of "Local Procedures"
```

Figure 7. The generated code

There really are only two embeds you need to be concerned about in the test procedure. One is the data embed, where you declare any data you need for the test. The other is Priority 5000, right before the return statement. That's where you place your test code.

Here's the code for the data embed:

```
detail  InvoiceDetail
```

And here's the code for the source embed:

```
detail.Init(12.50,3)
if detail.GetExtended() <> 37.50
   tr.message = 'Expected 37.50, got ' & detail.GetExtended()
else
   tr.passed = true
end
```

Save and try to compile the DLL. If you forgot to add the global extension to the DLL you'll get about 25 errors; if you remembered, you'll get something like seven errors.

Clearly there's a problem because you haven't yet defined the class!

The simplest way to create classes

There are a number of different ways you can declare classes in Clarion; some of them are fraught with danger. For instance, if you create classes that you export from a DLL so they can be used elsewhere, you had better get everything just right or you'll probably be looking at GPFs.

The approach I'll show you is the simplest, most foolproof way I know to create classes in Clarion and still reuse those classes everywhere you like.

First, add a new Clarion source file to your test DLL project (you can move it later). Now, adding a new source file to C7 via the solution explorer is pretty buggy.

Right-click on the *project* entry in the solution explorer (not the .APP itself). Choose Add | New Item, and pick any of the Clarion file types, it doesn't matter which.

C7 will add the file to the project tree, and it will open the file in the editor. But the file doesn't yet exist on disk. So you have to save it first, making sure you give it exactly the same name again.

One more thing - highlight the file in the solution explorer, and on the Properties pad change the Build Action from Compile to None. You don't want the project system compiling this file; it's going to be an INCLUDEd file.

With this new source file open (I called mine InvoiceDetail.clw) delete everything in the file and replace it with this code:

```
  Section('Header')
InvoiceDetail      Class,Type
GetExtended            procedure,real
Init               procedure(real price,long quantity)
                   End

  Section('Methods')
InvoiceDetail.GetExtended   PROCEDURE
  Code
  Return 0

InvoiceDetail.Init  procedure(real price,long quantity)
  Code
```

Now go back to the DLL APP file again. On the global embeds choose the After Global INCLUDEs embed point and insert the following source:

```
Include('InvoiceDetail.clw','Header')
```

Save the embed. Go to the Program Routines global embed and add the following source:

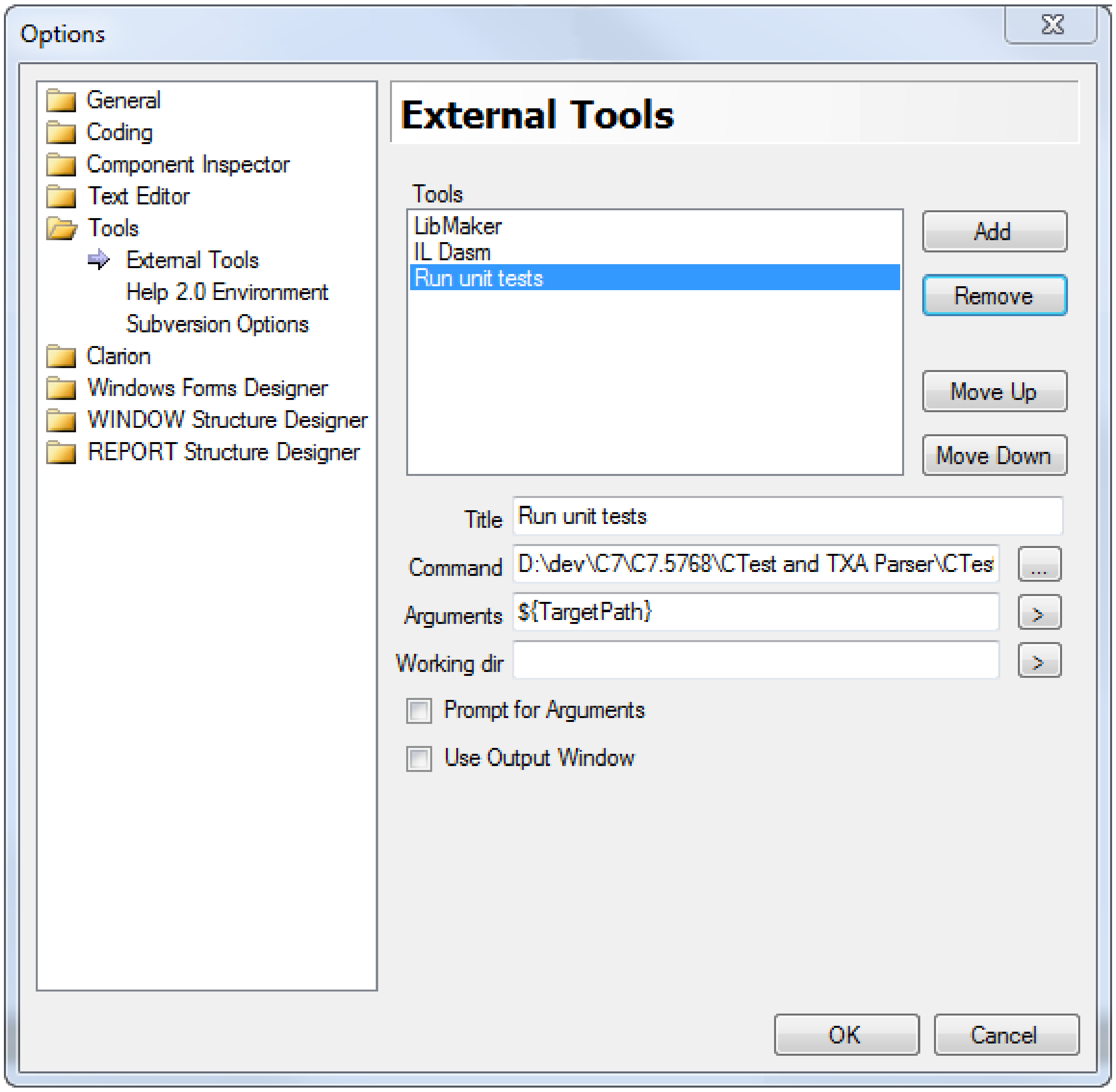
```
Include('InvoiceDetail.clw','Methods')
```

Notice that the Include statements both use a second parameter which corresponds to the Section heading in the source file. What you've done is include just the class header in the first embed, and the class methods in the second embed. You have to do this because it's a Clarion language requirement that class headers appear in the data section and class methods in the code section, and Program Routines is just a handy location in the program's code section.

You should now be able to compile the DLL.

Setting up CTest

There's one more thing you should do now to make unit testing as painless as possible, and that's to set up CTest in the C7 Tools menu. Choose Tools | Options, and from the list of Options choose Tools again. Click on Add to create an new external tool entry. Set the Command field to point to CTest.exe, and set the Arguments field to ${TargetPath} as in Figure 8.

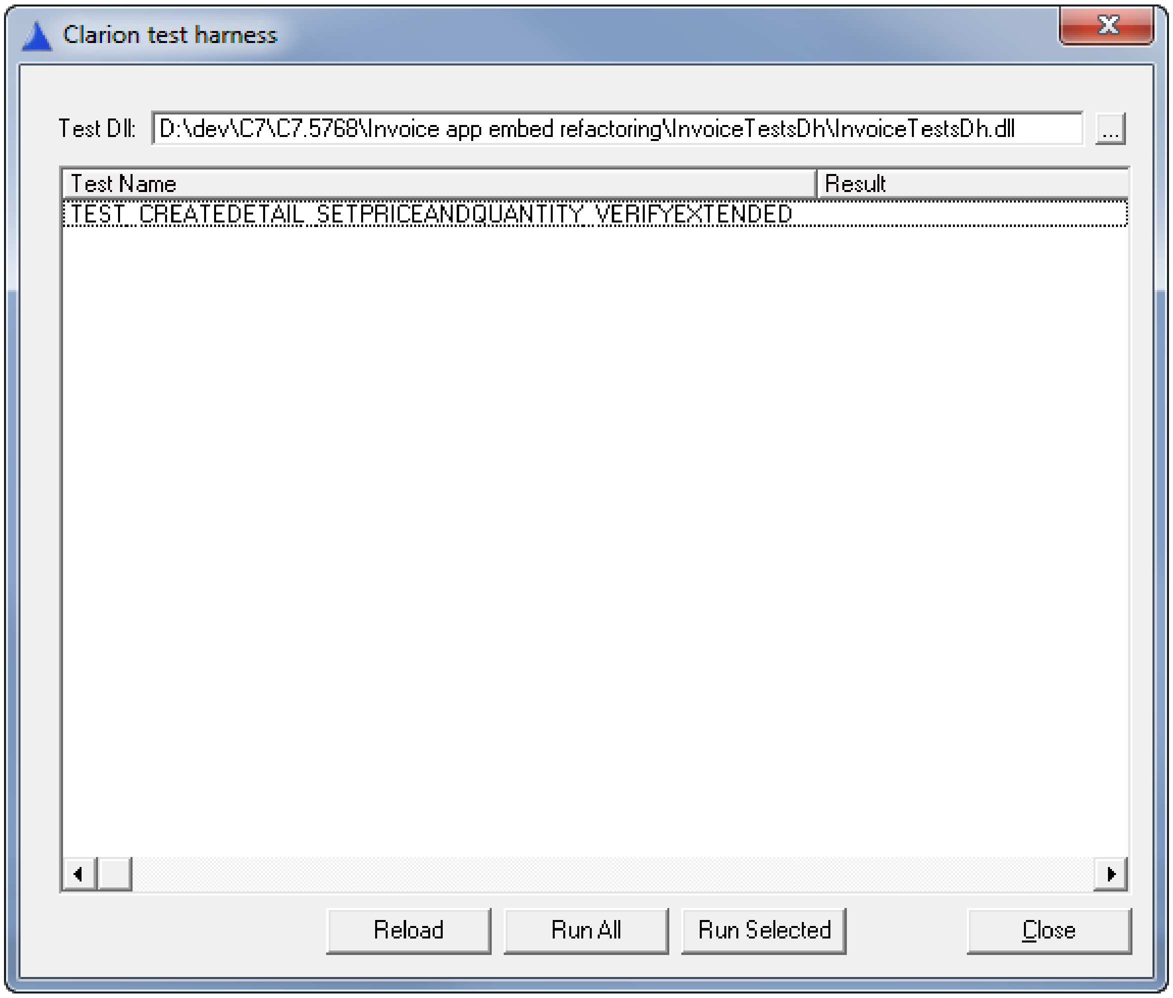


Tools:
- LibMaker
- IL Dasm
- Run unit tests

Title: Run unit tests
Command: D:\dev\C7\C7.5768\CTest and TXA Parser\CTes
Arguments: ${TargetPath}
Working dir:
[ ] Prompt for Arguments
[ ] Use Output Window

Figure 8. Setting up CTest as an external tool

With the test DLL as the active window in the IDE, choose Tools | Run unit tests. You should see the CTest window with the test DLL loaded, as in Figure 9.



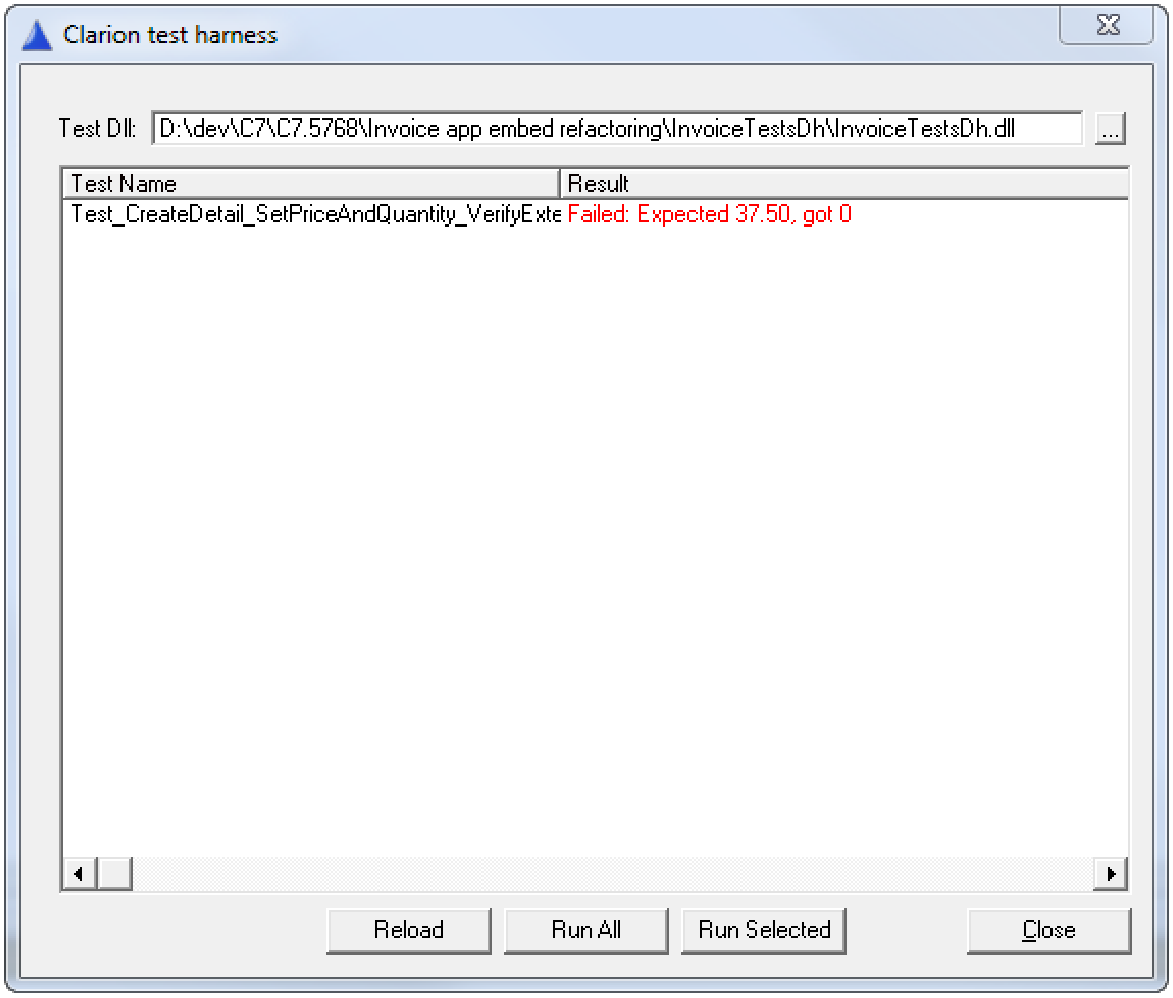
Test Dll: D:\dev\C7\C7.5768\Invoice app embed refactoring\InvoiceTestsDh\InvoiceTestsDh.dll

Test Name	Result
TEST_CREATEDETAIL_SETPRICEANDQUANTITY_VERIFYEXTENDED	

Figure 9. Loading the test DLL

You'll notice that the procedure name is in all caps - that's how the symbol is exported from the DLL. The procedure name is reformatted after the test is run, based on information returned by the test.

Press the Run All button. You should see a test failure, as in Figure 10.



D:\dev\C7\C7.5768\Invoice app embed refactoring\InvoiceTestsDh\InvoiceTestsDh.dll

Test Name	Result
Test_CreateDetail_SetPriceAndQuantity_VerifyExte	Failed: Expected 37.50, got 0

Figure 10. Test failure

GetExtended just returns zero. Change the class code to store the price and quantity in private variables, as follows:

```
    Section('Header')
InvoiceDetail      Class,Type
GetExtended            procedure,real
Init                procedure(real price,long quantity)
price               real,PRIVATE
quantity             long,PRIVATE
                 End

    Section('Methods')
InvoiceDetail.GetExtended   PROCEDURE
    code
    return self.price * self.quantity

InvoiceDetail.Init  procedure(real price,long quantity)
```

```
    CODE
    self.price = price
    self.quantity = quantity
```

Yes, I realize I'm using a real for the price - bear with me for a little while.

Compile the DLL.

If you exited the CTest application, restart it from the Tools menu. If you still have it up just press Reload. Then press Run All. The test should now pass.

Now, how about getting the discount amount? That involves two things: setting a discount percentage rate and calculating the discount. Really there are two things to test: Is the discount amount right, and is the total right? Let's start with the discount amount.

Create a new procedure called Test_CreateDetail_SetDiscount_VerifyDiscount. Select Test Procedure from the Template Types dialog, Defaults tab.

Or just highlight the first test, press Ctrl-C, then type in the new name. That will save a bit of setup typing.

Change the test code in this new procedure to

```
    detail.Init(12.50,3)
    detail.SetDiscountRate(10)
    if detail.GetDiscount() <> 3.75
      tr.message = 'Expected discount of 3.75, got ' & detail.GetDiscount()
    else
      tr.passed = TRUE
    end
```

Of course the test won't compile.

You'll get a number of unknown procedure errors field not found errors. That's because you don't have the SetDiscountRate, GetDiscount or GetTotal methods in the class. Add the declarations in the class header:

```
  GetDiscount          procedure,real
  SetDiscountRate        procedure(real discountRate)
```

And the implementation of the method (I've set GetDiscount to return 0 as a default):

```
  InvoiceDetail.GetDiscount          procedure
    code
    return 0



  InvoiceDetail.SetDiscountRate       procedure(real discountRate)
    code
```

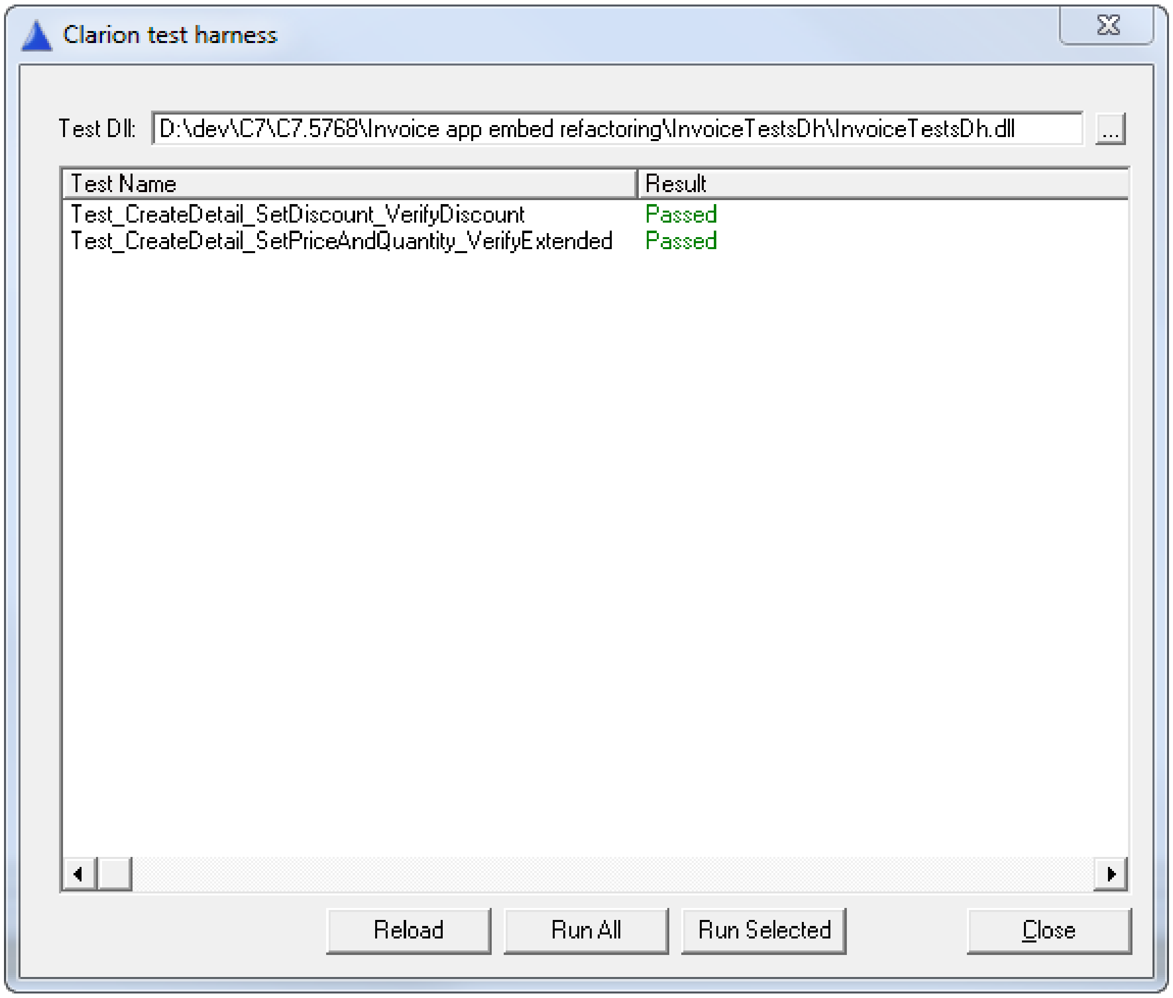
```
        self.discountRate = discountRate
```

Here's a little tip to making finding the compile error easier. If you've mistyped something or otherwise caused an error in the include file, do *not* (at least in 7.0) double-click on the error in the error list - the IDE will take you to program file that contains the Include statement. Instead just look in the include file. The error(s) will be highlighted.

Run the test - it fails, because there's no code to calculate the actual discount.

```
    InvoiceDetail.GetDiscount   PROCEDURE
        code
        return self.discountRate / 100 * self.GetExtended()
```

And what do you know - it works!



Test Dll: D:\dev\C7\C7.5768\Invoice app embed refactoring\InvoiceTestsDh\InvoiceTestsDh.dll

Test Name	Result
Test_CreateDetail_SetDiscount_VerifyDiscount	Passed
Test_CreateDetail_SetPriceAndQuantity_VerifyExtended	Passed

Figure 11. Two tests pass!

But wait - that result conveniently rounds out to a single decimal place. What happens if you use, say, a value of $11.45 and a discount of 11%? Plug that into your calculator and 3 * 11.45 * .11 = 3.7785. That's not an acceptable currency value. It should round up to 3.78. And in fact the test returns 3.7785, which is a failure.

You could Round() the result, but a better solution is simply not to use floating point numbers for financial calculations! Use decimal types instead. Here's an updated GetDiscount():

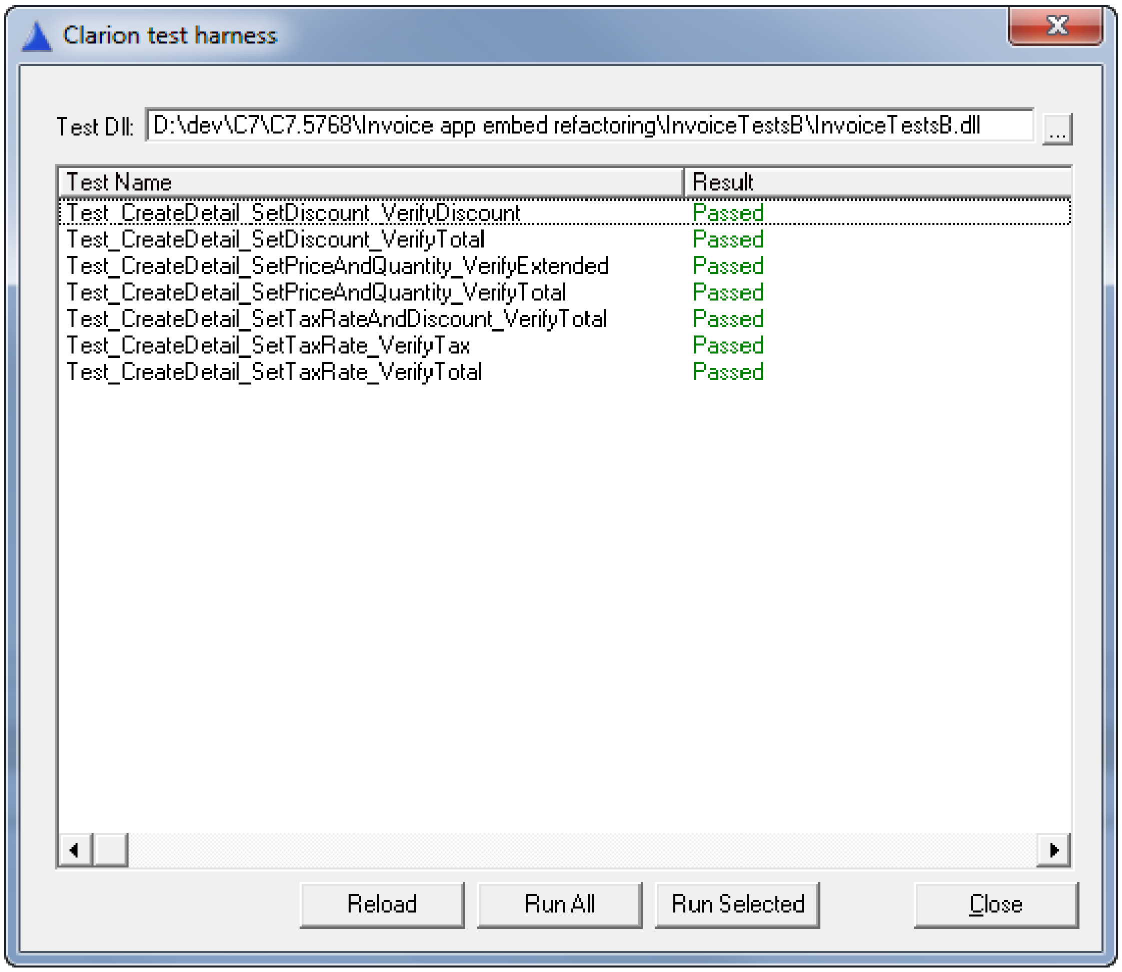
```
InvoiceDetail.GetDiscount   PROCEDURE
result    decimal(10,2)
    code
    result = self.discountRate / 100 * self.GetExtended()
    return result
```

I've also changed the DiscountRate variable to a Decimal(10,2). The Clarion Help file explains the rules under which the BCD (binary coded decimal) library is invoked - see Decimal Arithmetic in the index. The numeric constant (100) is a decimal, as is DiscountRate, and as long as one side of a mulitiplication is a decimal that will be a BCD operation as well (GetExtended returns a real). I store the resulting value in a temp decimal var for rounding purposes, and then return that value (again as a real).

But why don't I just use decimals entirely, instead of reals? Mainly because decimals have to be passed by address and can't otherwise be a return value. I find that a bit clunky. I'm not aware of any problems with passing the data as reals in this scenario, and if there are they should be uncovered by the unit tests ....

More tests

Figure 12 shows the remainder of the tests, which while not yet exhaustive do a reasonable job of covering the bases.



Test Dll: D:\dev\C7\C7.5768\Invoice app embed refactoring\InvoiceTestsB\InvoiceTestsB.dll

Test Name	Result
Test_CreateDetail_SetDiscount_VerifyDiscount	Passed
Test_CreateDetail_SetDiscount_VerifyTotal	Passed
Test_CreateDetail_SetPriceAndQuantity_VerifyExtended	Passed
Test_CreateDetail_SetPriceAndQuantity_VerifyTotal	Passed
Test_CreateDetail_SetTaxRateAndDiscount_VerifyTotal	Passed
Test_CreateDetail_SetTaxRate_VerifyTax	Passed
Test_CreateDetail_SetTaxRate_VerifyTotal	Passed

Figure 12. The test suite

You can find the code for the class in the downloadable source.

Implementing the class

Adding the class to the Invoice app is the same as adding it to the test DLL: just add the same two global include statements. And in the UpdateDetail procedure, add this line in the data section:

```
detail  InvoiceDetail
```

Then, in the CalcValues routine, replace the existing embed code with this code:

```
detail.Init(DTL:price,DTL:QuantityOrdered)
detail.SetTaxRate(DTL:TaxRate)
detail.SetDiscountRate(DTL:DiscountRate)
DTL:TaxPaid = detail.GetTax()
DTL:Discount = detail.GetDiscount()
DTL:TotalCost = detail.GetTotal()
DTL:Savings = detail.GetSavings()
```

You could, in fact, create an overloaded Init() method to take the tax and discount rates as well, and reduce this code from seven lines to five.

You'll also have to put the InvoiceDetail.clw file somewhere it can be found. That means either putting it in a directory that's already listed in the redirection file (such as the Clarion libsrc directory) or updating the redirection file. You may want to set up a source directory for just your own classes.

Summary

The original routine contained a hodgepodge of untestable, difficult to reuse code tied directly to a particular database.

The class that replaces that code isn't tied to any one database, can be reused easily without duplicating any business logic, and has an accompanying test suite to verify its behavior.

Extracting embed code into testable, reusable classes does involve some work, but the payoff is huge. You get verifiable code, and not just in the development phase. You can make changes to the class with much less risk of breaking existing code. If your unit tests are comprehensive, they'll catch any errors introduced by enhancements or even other bug fixes.

Tip: If you're using NetTalk, I'm told you need to add the "Suppress NetTalk" global extension in the test DLL.

Download the source

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

Converting Your Apps To C7

by Russell Eggen

Published 2009-12-21

When converting a C6 app to C7, you may get a message about a Window or Report structure having some issues. The procedures with such problems have a yellow icon next to their names.

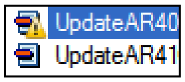


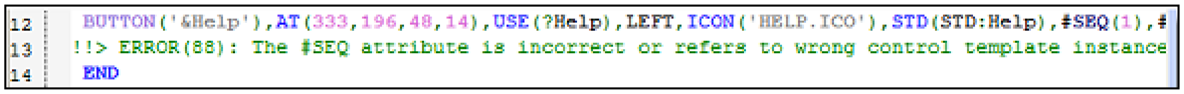
Figure 1. Procedure with warning icon

This article documents three such conditions and how to handle each. They are:

- Control template "missing" in C6 (simple fix)
- Control template persists after deletion in C6 (not so simple fix)
- Incorrect USE attribute

The Broken Control Template (simple fix)

If you open the window of the procedure C7 complains about, you may find a #SEQ attribute error (Figure 2).



```
 BUTTON('&Help'),AT(333,196,48,14),USE(?Help),LEFT,ICON('HELP.ICO'),STD(STD:Help),#SEQ(1),#
!!> ERROR(88): The #SEQ attribute is incorrect or refers to wrong control template instance
 END
```

Figure 2. A SEQ attribute error

The #SEQ number is actually correct, but what the error is really telling you that it can't find the rest of the template. Or to put it another way, a #SEQ attribute means a control template, but the rest of the template can't be found. There is really no elegant way of clearing this in C7 (you can, but its labor intensive).

One clue that something is wrong on the C6 side is shown in Figure 3.

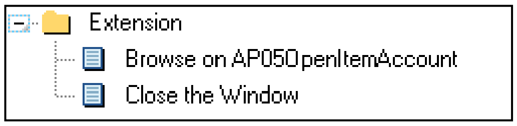
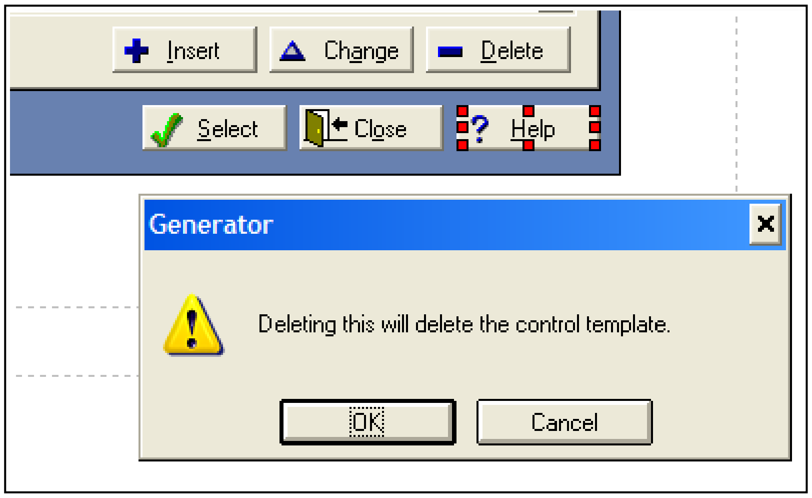


Figure 3.

There is supposed to be a "Call the Help" entry in that tree — it's missing.

Navigate to the same window as the C7 window shown above (with the error). Figure 4 shows the offending template that for some reason C6 has no issue with. Delete it as shown and press OK.



Deleting this will delete the control template.

Figure 4. Deleting the control template

Just repopulate it. The #SEQ will very likely be the same, but unlike the error exposed by C7, the rest of the template is now present. You can verify this by saving the window and looking at the same control/extension list from before (Figure 5).

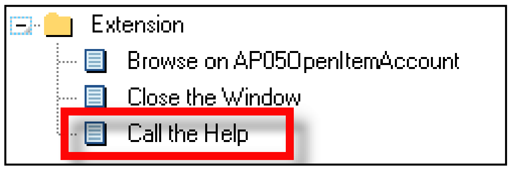
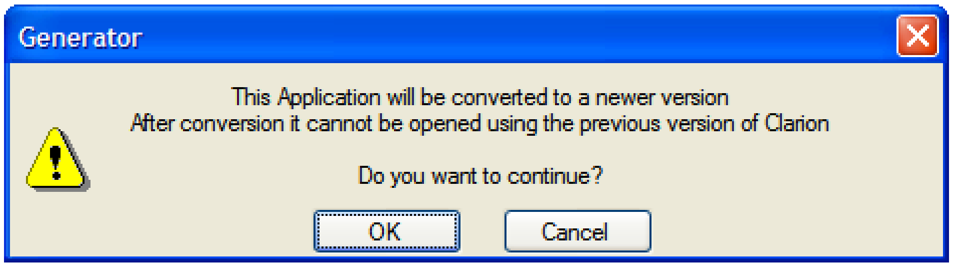


Figure 5. The Help template

Notice the control template is now *really* present? Copy the app to your C7 work area and convert it again (Figure 6).



This Application will be converted to a newer version
After conversion it cannot be opened using the previous version of Clarion

Do you want to continue?

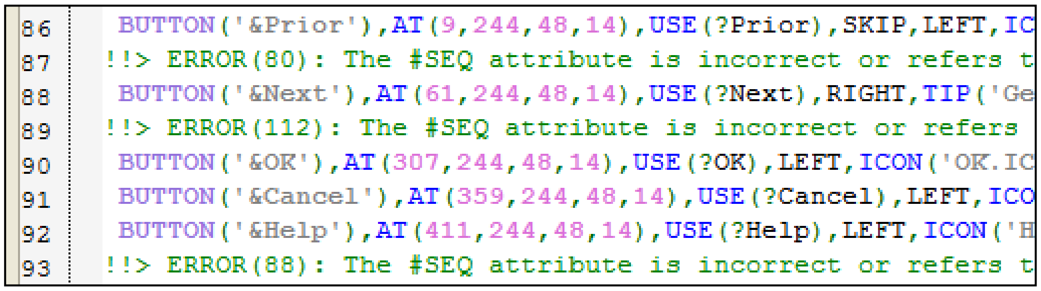
Figure 6. Reconverting the app

If you got all the issues C7 complained about, all the procedures in the app should be good to go. There could be issues later, but the generator/compiler/build process is a different topic.

The Broken Control Template (not so simple fix)

Not all control templates are built the same. Some can be very complex. Yet a simple template can produce complex issues. The following scenario happened with a custom control template I inherited. You may not run into anything like this situation, but I'm showing it here as a possible strategy.

Viewing the window source I saw the #SEQ errors in Figure 7 (as well as the help button issue as explained above).

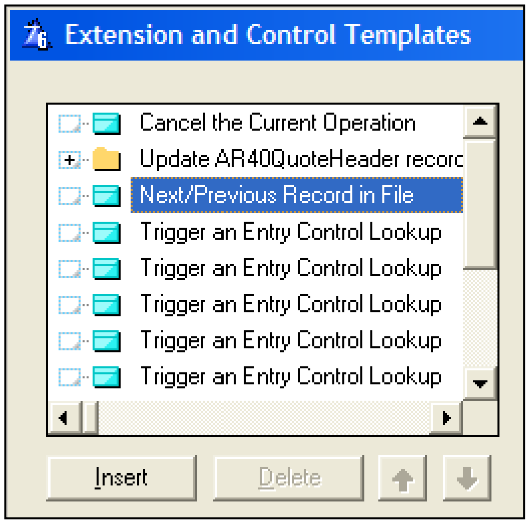


```
BUTTON('&Prior'),AT(9,244,48,14),USE(?Prior),SKIP,LEFT,IC
!!> ERROR(80): The #SEQ attribute is incorrect or refers t
BUTTON('&Next'),AT(61,244,48,14),USE(?Next),RIGHT,TIP('Ge
!!> ERROR(112): The #SEQ attribute is incorrect or refers
BUTTON('&OK'),AT(307,244,48,14),USE(?OK),LEFT,ICON('OK.IC
BUTTON('&Cancel'),AT(359,244,48,14),USE(?Cancel),LEFT,ICO
BUTTON('&Help'),AT(411,244,48,14),USE(?Help),LEFT,ICON('H
!!> ERROR(88): The #SEQ attribute is incorrect or refers t
```

Figure 7. SEQ errors

Switching back to C6 to correct this issue, I deleted the two buttons on the window as I did to fix the Help button issue. I saved and exited the window formatter.

Opening the extensions list I could see what C7 complained about in C6 (Figure 8).



Extension and Control Templates

- Cancel the Current Operation
- Update AR40QuoteHeader recorc
- Next/Previous Record in File
- Trigger an Entry Control Lookup
- Trigger an Entry Control Lookup
- Trigger an Entry Control Lookup
- Trigger an Entry Control Lookup
- Trigger an Entry Control Lookup

Figure 8. The control template is still there

The control template was still there! This was just the opposite of the first condition discussed in the prior section. No way could this application compile clean. Now what? Some light surgery was in order. It's actually very easy once you know the steps.

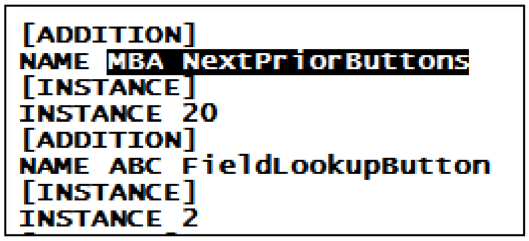
Step 1 — Export the procedure

In C6, choose File/Selective export. Choose a file name or accept the default. I find it easier to sort by name order so I select that tab. Then highlight the procedure to fix and either double click or press Select. Then press OK.

Step 2 — Edit the TXA

The TXA file is now in your C6 pick list, (select the source tab). Press Select. The source from the TXA is now open.

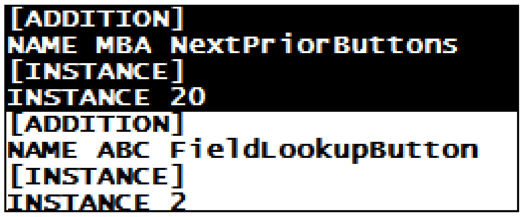
What you are searching for is the name of the extension, so use the search feature in the editor. Figure 9 shows the selected text.



```
[ADDITION]
NAME MBA NextPriorButtons
[INSTANCE]
INSTANCE 20
[ADDITION]
NAME ABC FieldLookupButton
[INSTANCE]
INSTANCE 2
```

Figure 9. Finding the extension

Starting with and including the [ADDITION] line, highlight everything until the next [ADDITION] line. It should look like Figure 10.



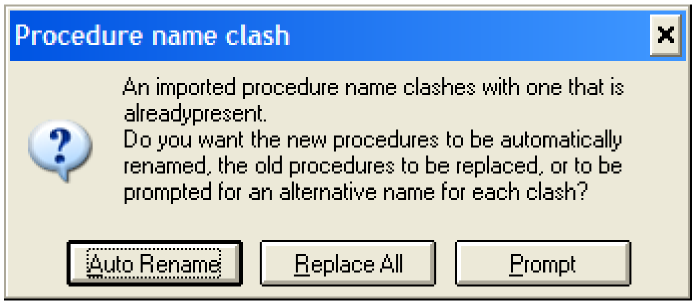
```
[ADDITION]
NAME MBA NextPriorButtons
[INSTANCE]
INSTANCE 20
[ADDITION]
NAME ABC FieldLookupButton
[INSTANCE]
INSTANCE 2
```

Figure 10. Deleting the extension

Press delete. If needed, press F3 to search again (which you would in the case of multiple procedures with the same condition). If you get a not found message, save and exit the TXA file.

Step 3 — Import the TXA

Choose File/Import from text. Select the name of the TXA edited from the prior steps. You get the procedure name clash message shown in Figure 11.



Procedure name clash

An imported procedure name clashes with one that is alreadypresent.
Do you want the new procedures to be automatically renamed, the old procedures to be replaced, or to be prompted for an alternative name for each clash?

Figure 11. Procedure name clash

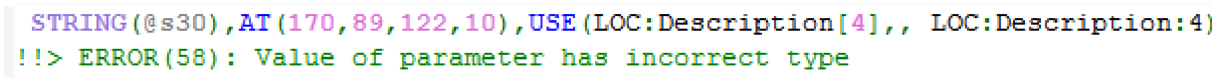
Press Replace All. The extra extension template listing should now be gone. Now simply add the control template on the window and it will be fine.

Save the C6 app and exit. Be sure the C7 version of this app is also closed.

Copy the C6 application to your C7 work area. Re-open the app (it's still in the solution). Hopefully you now find the app no longer has any errors.

The USE attribute

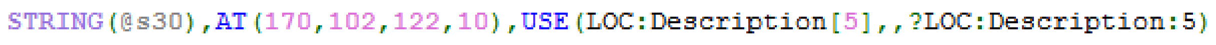
Sometimes you'll hit an issue dealing with the USE attribute in a window or report. Figure 12 is a screen shot of what I mean.



```
 STRING(@s30),AT(170,89,122,10),USE(LOC:Description[4],, LOC:Description:4)
!!> ERROR(58): Value of parameter has incorrect type
```

Figure 12. An invalid USE attribute

At first glance, this does not seem like there is anything wrong. Some may see it right away. Let me contrast the above with a similar control C7 does not have an issue with (Figure 13).



```
STRING(@s30),AT(170,102,122,10),USE(LOC:Description[5],,?LOC:Description:5)
```

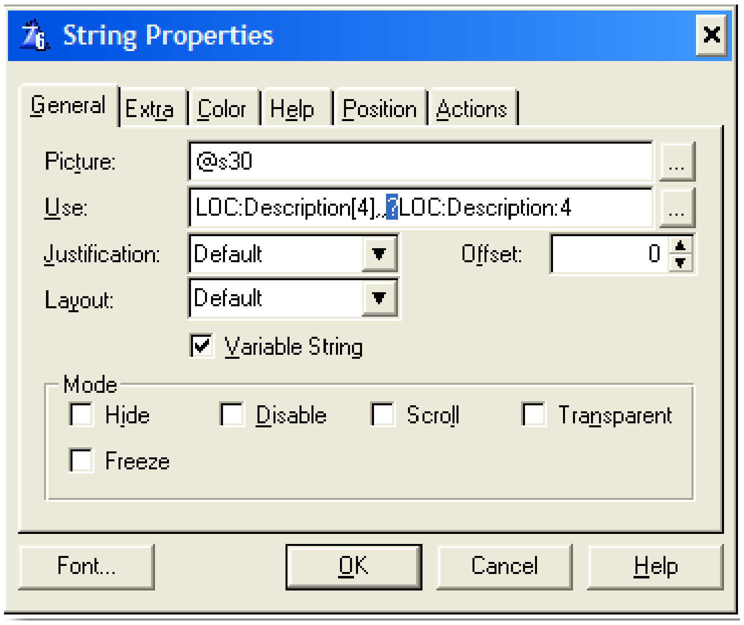
Figure 13. A valid USE attribute

See the difference (besides the different array number)? In the error window, the leading "?" mark is replaced by a space. This is easy enough to fix; just add back the missing question mark so it's a legal USE variable again.

You can test this by making that correction and save your changes. Open the window source again and if C7 is no longer complaining, it's fixed.

This fix happens entirely in C7, but to correct this in C6 you may use either the source view or the formatter and find the controls C7 has issues with and replace the missing question mark.

Figure 14 is the formatter view of the string properties of the same fix.



String Properties

General | Extra | Color | Help | Position | Actions

Picture: @s30

Use: LOC:Description[4],,?LOC:Description:4

Justification: Default

Offset: 0

Layout: Default

[✓] Variable String

Mode: [ ] Hide [ ] Disable [ ] Scroll [ ] Transparent [ ] Freeze

Figure 14. Fixing the USE attribute in the properties dialog

And if you use the property toolbox in the formatter, you can see the problem as well as implement the fix (Figure 15).

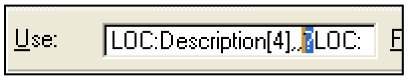

Use: LOC:Description[4],,?LOC:

Figure 15. Fixing the USE attribute in the property toolbox

Use whichever method you like the best. Repeat for all error conditions exposed by C7. If the developer copied this procedure and made changes to the copy, expect to find the same errors in the copied procedure.

Again, copy to your C7 work area and after converting again, C7 should be happy if you got all the errors.

If you have any other conversion tips, please post them as comments to this article.

Russ Eggen has been using Clarion since 1986. Until about 1996, he was using it for business applications, mostly accounting programs. Afterwards he joined Topspeed as a consultant, and later as an instructor. He was a founding member of SoftVelocity when that company formed from Topspeed in May 2000. He left SoftVelocity in January 2001 and now works for his own company, RadFusion Inc. He still teaches and lectures, and is currently working on a new book and setting up a local Clarion classroom. Russ enjoys flying, scuba, and applied philosophy, and with great effort you might coax him into political discussions.

Reader Comments

Add a comment

Clarion Magazine

SQL Tips and Tricks

by Joe Tailleur

Published 2009-12-23

Clarion is an amazingly productive environment, with its rich templates and third party developers. You can very quickly create some amazingly useful and complex programs right out of the box, and development times are really quick.

Clarion does however have a weakness. Although TPS files are an excellent platform for many projects, especially when the number of users is limited, once you step past a certain number of users (my limit is five), or if you are working within a particular environment (remote connections, WAN, slow network, etc.), those limitations can quickly show themselves, resulting in reliability issues and reduced speed.

Although I work with both MySQL and MS-SQL, my personal choice for any SQL project is Microsoft SQL server.

The reasons I choose MS-SQL are simple:

- It is one of the best performing and reliable SQL engines currently available
- Clarion has a native MS-SQL driver built in. No ODBC to set up or configure on the workstations.
- It scales incredibly well. Reasonably well written programs accessing a MS-SQL backend run incredibly well, even with millions of records.
- It tunes itself as it goes. It'll create indexes for you automatically, has full text search capabilities, and built in, easy to use "Maintenance" functions that allow you to very easily and quickly create automated backup processes for your very valuable data.

Using Clarion with SQL

To successfully use Clarion with SQL, you really need to change how you think about data.

Clarion uses a "record-by-record" approach that works incredibly well with page-loaded lists and a relatively limited data set.

SQL uses a "results" based approach. You send commands to the server, telling it what you want, and then you process the results of that query.

In this article I'll cover some of the techniques I use to get the most out my SQL server, including:

- PROP:SQL and the dummy table
- Faster reports with SQL and the In-Memory Database Driver (IMDD)
- Deleting records with PROP:SQL
- Filtering
- Self-joins
- SQL totals

I'll assume you have some basic knowledge of SQL (check out the SQL entry in the topical index for more articles).

Field Names

I personally always try to add an external name to any Clarion field (Column) used in an SQL table. I use a PREFIX_FieldName format for my external name. This is to make sure each field (even across tables) has a unique name. When you are building complex queries, this can become a factor. In fact, some flavours of SQL require each field to have a unique name.

The other very good reason to use PREFIX_FieldName is that it practically guarantees you won't run into reserved words.

PROP:SQL and the Dummy table

PROP:SQL can do some amazing things, but it can be a difficult concept to grasp at first. The short explanation is that it lets you send your own custom SQL statements directly to the server. The problem is you need some way to get information back from the server. One common way to do this in Clarion is with an SQL "dummy table". If you're not familiar with dummy tables you can read Steve Parker's article on the subject.

There are a couple of things that I'd like to talk about.

- Although you do NOT need to use a "sqldummy" table to hold results, I do personally like using the dummy table as it helps to keep things neat, readable, and safer, especially when separately processing results issuing multiple PROP:SQL statements.. You can actually issue PROP:SQL statements using any SQL table (I'll touch on that some more shortly).

- PROP:SQL can make a difference of unbelievable proportion. Some processes will literally run 100's of times faster using PROP:SQL statements.

One thing to take note of: In order to use an SQL "dummy" table, the table doesn't actually need to exist on the server, just in the Clarion dictionary.

Create a dummy SQL table

Here's how to create a dummy table:

- I typically create a table called sqlFile in my dictionary, with STRING(255) fields called F1, F2, etc. Depending on the application, you many just need a few or many. You will need as many "Fields" here as the number of fields you retrieve during your queries. I normally just start with 10 and add them as I need them. It's not really a table, and never gets created, so there will be no "Error 47" problems.
- In the table Driver Options put /TURBOSQL=True.
- If you are using FM3, also add FM3IGNORE as a user defined option.
- Clear the Create Table checkmark

Once you've created your dummy SQL table in the dictionary, you are ready to start using PROP:SQL statements.

I use SQL code typically in three ways.

- Reports (PROP:SQL)
- Browses (BRW1.SetFilter('SQL'))
- Maintenance tasks (like purging old records to history tables).

Reports using an In-memory table to hold results

This is an example from a report I run that gathers information from two tables, stores the results in an in-memory table, then prints the report. The nice thing about this, is using a join to a "Status" table, I can filter the result set before it gets to me. For example, Sold Vehicles don't appear.

What I normally do is call a source procedure first, which then gathers the information and fills the reports table, and then prints the report. You can also do the same thing from within the report, inside an early embed (just after OPENFILES in the Init section). I chose to use the source approach because it lets me use the same report for more than one purpose. I can set up a standard report format, and then simply fill it with different data using different source procedures with a header variable to change the name of the report.

Here is the source procedure:

```
Do OpenFiles

Empty(Reports)      ! This is my memory tableo

! New Trucks
Do ClearSqlBuffer   ! Empty the dummy table
loc_Header = 'NEW TRUCKS'
sqlFile{PROP:SQL} = ' SELECT TRU_SERIALNUMBER' |
   & ',TRU_UNITNUMBER' |
   & ',TRU_YEAR' |
   & ',TRU_MODEL' |
   & ',TRU_FTLINVOICEDATE' |
   & ',TRU_FTLCOST' |
   & ',TRU_STATUS' |
   & ',TRU_Make' |
   & ' FROM NewTrucks, Status' |
   & ' Where TRU_STATUS = STA_STATUS and STA_PRINTYN = 1'
Do PostResults

! New Trailers
loc_Header = 'NEW TRAILERS'
Do ClearSqlBuffer
```

```
    sqlFile{PROP:SQL}    = 'SELECT TRA_SERIALNUMBER' |
        & ',NEWTRL_UNITNUMBER' |
        & ',NEWTRL_YEAR' |
        & ',NEWTRL_MODEL' |
        & ',NEWTRL_DATERECEIVED' |
        & ',NEWTRL_INITIALCOST' |
        & ',NEWTRL_STATUS' |
        & ',NEWTRL_ MAKE ' |
        & 'FROM NewTrailers, Status ' |
        & 'Where NEWTRL_Status = STA_STATUS and STA_PRINTYN = 1'
    Do PostResults
    PrintNewFloorPlan
    Do CloseFiles
```

I also have two routines:

```
 ! This routine guarantees that the buffer is
 ! clear between queries.

 ClearSQLBuffer  ROUTINE
    Access:sqlFile.Close
    Access:sqlFile.Open
    Access:sqlFile.UseFile
    Clear(sqlfile:record)



 ! This routine adds the buffer contents to the
 ! Reports table (In-Memory)

 PostResults Routine
```

```
Loop Until Access:sqlFile.Next()
   REP:Field1  = sqlfile:F1          ! Serial Number
   REP:Field2  = sqlfile:F2          ! Unit Number
   REP:Field3  = sqlfile:F3          ! Year
   REP:Field4  = sqlfile:F4          ! Model
   REP:Field5  = sqlfile:F5          ! Invoice Date
   REP:Field6  = sqlfile:F6          ! Cost
   REP:Field7  = sqlfile:F7          ! Status
   If REP:Field5 <> 0
      REP:Field8  = Today() - sqlfile:F5  ! Days
   Else
      REP:Field8 = 0
   End
   REP:Field9  = loc_Header
   REP:Field10 = sqlfile:F8
   Access:Reports.Insert()
End
```

Another little discovery I made was when printing child records, and using the TakeRecord embed, filtering out the details and then manually processing the child records (see Steve Parker's article on this subject).

Here is my original embed code:

```
! Print Consignments
loc_Header = 'TRUCK CONSIGNMENTS - ' & Clip(LOC:Location)
Print(RPT:InventoryHeader)
CONSIG:Location = LOC:Location
CONSIG:UnitNumber = 0
SET(CONSIG:LocationKey, CONSIG:LocationKey)
Loop Until Access:Consignment.Next() <> Level:Benign
   If CONSIG:Location = LOC:Location
```

```
        STA:Status = CONSIG:Status
        Access:Status.Fetch(STA:StautsKey)
        If Sta:Status = CONSIG:Status and STA:PrintYN = '1'
          loc_UnitNumber  = CONSIG:UnitNumber
          loc_CurrentCost = CONSIG:CurrentCost
          loc_SerialNo    = CONSIG:SerialNumber
          loc_Year        = CONSIG:Year
          loc_Make        = CONSIG:Make
          loc_Model       = CONSIG:Model
          lOC_Colour      = CONSIG:ExteriorColour
          loc_Status      = CONSIG:Status
          Print(RPT:Inventory)
        End
      Else
        Break
      End
    End

    Print(RPT:Space)
    loc_Header = 'TRAILER CONSIGNMENTS - ' & Clip(LOC:Location)
    Print(RPT:InventoryHeader)
    CONTRL:Location = LOC:Location
    CONTRL:UnitNumber = 0
    Set(CONTRL:LocationKey, CONTRL:LocationKey)
    Loop Until Access:ConsignmentTrailers.Next() <> Level:Benign
      If CONTRL:Location = LOC:Location
        STA:Status = CONSIG:Status
        Access:Status.Fetch(STA:StautsKey)
        If Sta:Status = CONTRL:Status and STA:PrintYN = '1'
          loc_UnitNumber  = CONTRL:UnitNumber
```

```
            loc_CurrentCost = CONTRL:CurrentCost
            loc_SerialNo    = CONTRL:SerialNumber
            loc_Year        = CONTRL:Year
            loc_Make        = CONTRL:Make
            loc_Model       = CONTRL:Model
            lOC_Colour      = CONTRL:PAINT
            loc_Status      = CONTRL:Status
            Print(RPT:Inventory)
          End
        Else
          Break
        End
      End
      Print(RPT:Space)
```

This report took approximately three to five minutes to run locally (the server is in my building), and almost 20 minutes to run via the WAN.

I replaced the above code with the following SQL style code:

```
      ! Print Consignments
      loc_Header = 'TRUCK CONSIGNMENTS - ' & Clip(LOC:Location)
      Print(RPT:InventoryHeader)
      do ClearSQLBuffer
      sqlFile{PROP:SQL} = 'SELECT' |
          & ',CONSIG_UnitNumber' |
          & ',CONSIG_CurrentCost' |
          & ',CONSIG_SerialNumber' |
          & ',CONSIG_Year' |
          & ',CONSIG_Make' |
          & ',CONSIG_Model' |
          & ',CONSIG_ExteriorColour' |
          & ',CONSIG_Status' |
          & ' FROM dbo.Consignment, dbo.Status' |
          & ' WHERE CONSIG_Status = STA_Status' |
          & ' AND STA_PrintYN = 1 ' |
          & ' AND CONSIG_Location = <39>' & LOC:Location & '<39>'
      Do PrintResults
      Print(RPT:Space)

      loc_Header = 'TRAILER CONSIGNMENTS - ' & Clip(LOC:Location)
      Print(RPT:InventoryHeader)
      do ClearSQLBuffer
      sqlFile{PROP:SQL} = 'SELECT ' |
          & ' CONTRL_UnitNumber' |
          & ',CONTRL_CurrentCost' |
          & ',CONTRL_SerialNumber' |
          & ',CONTRL_Year' |
          & ',CONTRL_Make' |
          & ',CONTRL_Model' |
          & ',CONTRL_PAINT' |
          & ',CONTRL_Status' |
          & ' FROM dbo.ConsignmentTrailers, dbo.Status' |
          & ' WHERE CONTRL_Status = STA_Status' |
          & ' AND STA_PrintYN = 1 ' |
          & ' AND CONTRL_Location = <39>' & LOC:Location & '<39>'
      Do PrintResults
      Print(RPT:Space)

    PrintResults    ROUTINE
    Loop Until Access:sqlFile.Next() <> Level:Benign
      loc_UnitNumber     = sqlfile:F1
      loc_CurrentCost    = sqlfile:F2
      loc_SerialNo       = sqlfile:F3
      loc_Year           = sqlfile:F4
      loc_Make           = sqlfile:F5
      loc_Model          = sqlfile:F6
      lOC_Colour         = sqlfile:F7
      loc_Status         = sqlfile:F8
      Print(RPT:Inventory)
    End
```

The SQL style coded report prints in under five seconds, locally or across the WAN. I did not use an in-memory table for this report, just local variables in the report.

Several things come into play here that affect the speed. Clarion always fetches the full record when you loop through a file, even if that file is an SQL table. These tables are quite large, containing approximately 75 fields. Using SQL code, I am only fetching eight of them, and I do not have to also fetch a record from the Status table as well each loop through.

Deleting Records using PROP:SQL

Weekly I import the inventory tables from our accounting system for use in our in-house Purchase system. The inventory is set up so each of our locations has its own set of parts. There is quite a bit of overlap in part numbers, but there are also a lot of things (sales history, purchase history, costs, etc.) that are unique to each branch.

So before doing the import of the new information, I delete the old location's inventory records. This used to involve a LOOP and a Delete statement (I can't Empty the table, as everyone else's inventory information is in there). To do the delete used to take several minutes of looping through the records and deleting each item by location_ID.

Now, using SQL code, it takes a couple of seconds.

```
  sqlFile{PROP:SQL} = 'DELETE FROM dbo.PartsLookup' |
    & ' Where PAR_Location_ID = ' & loc_Location_ID
```

loc_Location_ID is a Clarion variable from a window.

Test your deletes!

To test a delete statement without risking unwanted deletes, replace Delete with Select * and run it through your SQL query analyzer. So the above query would become:

```
! Need to put an actual ID in to test the
! query in the analyzer.
SELECT  * FROM  dbo.PartsLookup
    WHERE  PAR_Location_ID = 4
```

This gives you a chance to test your logic before committing to actually removing the records.

Qualifying deletions

You can only delete from a single table at a time, but you can use joins or sub queries to qualify your deletion:

```
sqlFile{PROP:SQL} = 'DELETE from PurchaseItems' |
         & ' WHERE POI_Purchase_ID IN' |
         & ' (SELECT PUR_Purchase_ID FROM Purchase' |
         & ' WHERE PUR_PurchaseDate < ' & Today() - 270 & ')'
```

This deletes items from the PurchaseItems table, where the purchase date of the Purchase table (the parent) record is older than a certain date.

Filtering browses

You can use SetFilter in combination with the SQL statement to speed up browse filtering. While this is not a PROP:SQL type statement, its performance affects can be as drastic, and you gain the amazing flexibility afforded by SQL statements.

```
BRW1.SetFilter('SQL(FIN_NAME LIKE <39>%' & Clip(loc_Search) & '%<39>)')
```

While the above is a pretty simple example, you can do amazingly complex queries in your filter, including the use of *sub queries*.

```
BRW1.SetFilter('SQL( (CUS_CustomerName LIKE <39>%' |
 & Clip(loc_Search) & '%<39> OR' |
 & ' CUS_Name2 LIKE <39>%' & Clip(loc_Search) & '%<39> OR' |
 & ' CUS_CompanyName LIKE <39>%' & Clip(loc_Search) & '%<39> OR' |
 & ' CUS_FirstName LIKE <39>%' & Clip(loc_Search) & '%<39> OR' |
 & ' CUS_LastName LIKE <39>%' & Clip(loc_Search) & '%<39> OR' |
 & ' Exists (select * from dbo.CustomerContacts' |
 & ' WHERE CUT_CustomerID = CUS_CustomerID AND' |
 & ' (CUT_FirstName LIKE <39>%' & Clip(loc_Search) & '%<39> OR' |
 & ' CUT_LastName LIKE <39>%' & Clip(loc_Search) & '%<39>))))')
```

The example above searches not only the main customer table, but also the contacts table for our search variable.

Self joins

Another concept I really like is the *self join*:

```
SELECT A1.PAR_PartNumber,
   A1.PAR_Description,
   A1.PAR_MonthsNoSale,
   A1.PAR_Cost,
   A1.PAR_OnHandQty,
   C1.LOC_Location,
   B1.PAR_YearlySales,
   D1.LOC_Location
 FROM PartsLookup AS A1, PartsLookup AS B1,
   Locations AS C1, Locations AS D1
```

```
WHERE A1.PAR_Location_ID = 4
  AND C1.LOC_Location_ID = A1.PAR_Location_ID
  AND D1.LOC_Location_ID = B1.PAR_Location_ID
  AND A1.PAR_PartNumber = B1.PAR_PartNumber
  AND B1.PAR_YearlySales > 0
  AND A1.PAR_Obsolete = 1
  AND B1.PAR_Obsolete <> 1
  AND A1.PAR_OnHandQty > 0
  AND B1.PAR_MonthsNoReceipt < 24
ORDER BY A1.PAR_MonthsNoSale DESC, B1.PAR_YearlySales DESC
```

This particular query gives me a list of obsolete parts from Location 4 that have activity in other locations (all data is inside the same table).

I am creating a join to the Locations table in order to display a location name, as opposed to just the ID field.

The key here is the use of SQL aliases, which are much like Clarion ALIAS files. PartsLookup is declared twice, first as

```
PartsLookup AS A1
```

and then as

```
PartsLookup AS B1
```

Only obsolete parts that are active in other locations are displayed. I use this query to tell me where I might be better off allocating our inventory.

SQL Totals

Totaling with SQL is super quick and easy! I normally create a routine that I call from a ResetFromAsk embed or anywhere else I might want to calculate my totals.

```
CalculateTotals ROUTINE
```

```
sqlfile{PROP:SQL} = 'select sum(POD_TOTALCOST)' |
    & ',count(POD_TOTALCOST)' |
    & ',sum(POD_TOTALSELL)' |
    & ' FROM PurDetails' |
    & ' WHERE POD_PONUMBER = ' & PUR:PoNumber
Next(sqlfile)
PUR:TotalCost = sqlfile.F1
loc_Lines = sqlfile.F2
PUR:TotalSell = sqlfile.F3
```

You'll notice here that you don't need to loop through the results. The PROP:SQL statement will only return a single set of results, so simply using next(sqlFile) gets the information.

You can also use calculations in your PROP:SQL code. Here I'm using the SUM function:

```
sqlFile{PROP:SQL} = 'SELECT TOP 20' |
  & ' PAR_Location_ID' |
  & ',PAR_PartNumber' |
  & ',PAR_Description' |
  & ',SUM(PAR_COST * PAR_ExcessQty) AS Total' |
  & ',SUM(PAR_ExcessQty) AS TotalQty' |
  & ',PAR_COST' |
  & ',PAR_DateAdded' |
  & ',PAR_MonthsNoSale' |
  & ',PAR_YearlySales' |
  & ',PAR_OnHandQty' |
  & ' FROM dbo.PartsLookup' |
  & ' WHERE PAR_Location_ID = ' & LOC:Location_ID |
  & ' GROUP BY PAR_Location_ID' |
  & ',PAR_PartNumber' |
  & ',PAR_Description' |
  & ',PAR_Cost' |
  & ',PAR_DateAdded' |
  & ',PAR_MonthsNoSale' |
  & ',PAR_YearlySales' |
  & ',PAR_OnHandQty' |
  & ' ORDER BY Total DESC'

Loop Until Access:sqlFile.Next() <> Level:Benign
   REP:Field2 = LOC:Location       ! Location
   REP:Field3 = sqlfile:F2        ! Part Number
   REP:Field4 = sqlfile:F3        ! Description
   REP:Field9 = sqlfile:F4        ! Excess Value
   REP:Field5 = sqlfile:F5        ! Excess Qty
   REP:Field6 = sqlfile:F6        ! Cost
   REP:Field7 = sqlfile:F7        ! Date Added
   REP:Field8 = sqlfile:F8        ! MNS
   REP:Field10 = sqlfile:F9        ! Yearly Sales
   REP:Field15 = sqlfile:F10       ! On Hand Qty
   Access:Reports.Insert()
End
```

Converting to SQL: it's easier than you think

Some applications can be converted to SQL by simply changing your dictionary and recompiling, especially if you are already using a unique key in your tables. But you'll need to make sure you're using SQL-compatible data types. See the conversion articles by Scott Ferrett and Stephen Mull for more.

Although there are many options for optimizing your application, getting started can really happen in a very short amount of time, and most of your applications will run just fine using pure Clarion code.

Switching some of your queries though to SQL specific code can have a huge impact on how your program runs. Get started, and then slowly make your way through your application to optimize your code.

Joe Tailleur picked up Clarion 2.1 CPD in 1992 after dabbling in DBase IV. He was looking for something to write an Inventory control application and wanted to compile to .EXE and not have to worry about run time or licensing fees. Once he got into Clarion he was amazed at how much better it was than anything he'd used previously. An IT manager since 2002, he moved into a SQL environment in 2005 to overcome performance issues with TPS files accessed across a WAN. Joe lives in Northern British Columbia, Canada, and has a passion for things mechanical. In his off time, his summer days are filled with motorcycles or bicycles (road, mountain, or downhill riding) and winters with snowmobiling. He is also an avid sports fan (football mostly) and has played hockey and football for quite a few years. He and his wife have been married for 21 years and have a teenage son who also is a bit of a computer junkie. Joe's best days are when he learns or discovers something or somewhere new.

Reader Comments

Posted on Thursday, December 24, 2009 by Arturo Rivera

Easy to learn, thank's for the tips.

Posted on Sunday, December 27, 2009 by Wim Steur

Perfect supplement on your Clarion Live webinar. Lots of useable tips which I didn't know and I'm going to use.

Thanks Joe!

Add a comment

Clarion Magazine

First Look: Clarion 7.1

by Dave Harms

Published 2009-12-31

It's a full six months since the last release of Clarion 7 (build 5768). That build was the third post-gold release sent out to CSP subscribers.

Why such a long wait? I expect it had a lot to do with the largely negative assessment given 7.0 by the Clarion developer community. There were way too many bugs, and still some important features missing from the product, including an accurate window previewer and a report writer.

The C7 product team has clearly been busy in the intervening half year. I'll list the major changes in 7.l first (largely following the release notes) and then I'll discuss some of the reactions among Clarion developers.

Clarion 7.1 changes

As with any major release, C7.1 includes a number of bug fixes. For the most part they aren't reflected here; rather, this is a short list of fixes and new features. You can get a more complete list here.

- An expanded command line interface to the IDE
- The IDE now writes out a recovery log, so in the event of a crash you can recover changes that would otherwise be lost. As well, the "most recent" list is preserved in the event of a crash.
- TPSFix supports enhanced encryption.
- There is an initial release of a new Property Grid. Features include: allow tab key navigation, display checkboxes for true/false, spin box for integers, undo/redo, auto-expand child properties, etc.
- Subversion support.
- No more window/report source view, unless you really want it.
- A new Save and Exit button on the AppGen window makes it more obvious when you're about to close the App.
- A new window previewer.
- In the Data/Tables Pad and the dictionary editor you can now paste in field definitions copied from source code. When copying and pasting the details of a field that has an initial value from the Dictionary Editor or the Data / Tables Pad, the definition now includes the default value where possible.
- Option to automatically save dictionary changes as when modifying columns. Previously you were asked each time if you wanted to save your changes.
- Improved support for handling Microsoft SQL Server 2008 data types.
- New XML examples
- New property syntax statements, including one for last-ditch error handling.
- The template language adds support for pre and post build events using the #PROJECT statement.
- A beta release of the new ReportWriter. This report writer is built on top of the DevExpress Report Writer, which is a .NET

product. I'll have a separate review of the new ReportWriter in the near future.

Clarion 7.1 in the wild

Shortly after 7.1 became available I posted a survey question: "In your opinion, is Clarion 7.1 ready to fully replace Clarion 6.x?"

So far, about 40% of respondents think there's at least a possibility that 7.1 is solid enough to replace 6.x, although only 5% are absolutely sure of this. And there have been a number of new bug reports and issues raised, as you'd expect. On the whole, however, I think the response has been significantly more positive than for 7.0.

Keyboard shortcuts are still a bit of a problem (although as noted on ClarionFAQ, you can at least customize these). For one thing, Ctrl-Shift-M doesn't seem to trigger a Generate and Make (via Application Pad settings), as it should. I've also noticed that I get up to several seconds of disk thrashing whenever I do a Generate. I also see the same kind of thrashing whenever I return from a procedure, and each time the .AP~ file grows by about 300K. C7.0 doesn't have this same behavior, so perhaps it has something to do with crash recovery. See PTSS 34936 if you have access to the bug tracking system.

There's still no support in 7.1 for PNG image files, which seems like a glaring omission.

Owner drawn menus remain a very contentious point. Prior to Clarion 7, menus were operating system menus, which meant they could be skinned by products like the Noyantis templates. In Clarion 7, the Clarion RTL takes control over the menu's appearance, and the result seems to be pleasing just about no one. SoftVelocity has until recently resisted any change to the new menus, but recently Bob Zaunere mentioned that they would consider a switch to let the developer decide which approach to use (OS or owner drawn).

Some developers have noticed problems with Norton's anti-virus product, specifically the Sonar option which apparently sees the C7 executables as infected with a virus (a false positive) and deletes them from your Clarion bin directory. I'm told you can disable Sonar without having to get rid of Norton entirely.

Based on other reports, you may get occasional lockups in 7.1, particularly when importing C6 apps, although the IDE seems much more stable than 7.0. If you run into problems getting a clean compile, try deleting all the generated source files and compiled OBJ files and rebuilding.

A few developers have had difficulty installing C7.1, although for most this has been uneventful. The usual install culprit is the Visual Studio VC++ runtime, which the install program will supply. For some reason this part of the install occasionally fails or fails to trigger, in which case you have to install the runtime manually. The trick is that even on 64 bit Windows you have to install the 32 bit runtime, since the IDE is 32 bits. According to Bob Foreman, you should uninstall the VC++ 2005 runtime distributable on your machine, then download and install this one.

You may have a problem with the properties not showing up when you edit a Window or Report structure. You can try going to Tools | Options | Appearance and changing your preferred ambience from nothing to Clarion. If that still doesn't work, try deleting all of the application/solution CLW and INC files and doing a generate and make. This bug is reported fixed for the next release.

There's definitely some frustration out there over minor usability issues that haven't been addressed. If you go to an embed list and choose just the filled embeds (which by the way shows same-level embeds that aren't filled), your setting isn't remembered. The next time you go to the embed list you see all the embeds again.

Some developers blame usability issues on the IDE (that is, the SharpDevelop code base); others find the IDE to be pretty solid, and complain about the Clarion-specific aspects. And sometimes the integration just isn't tight enough. For instance, menu focus is still a problem with the AppGen. Open a submenu on the main IDE menu, then click on an open APP, and the open menu doesn't go away as it should. It may be that the AppGen is Win32 code, and there's a problem with focus as a result, but these things should be addressed.

In the "you shouldn't have been doing that in the first place" department, it's been noted that a class declared at the module level no longer has access to a class declared at the procedure level. This is something the C6 compiler allows, and shouldn't. According to Diego Borojovich, EasyListView and EasyNavBar are affected by this issue, although I haven't confirmed that.

Another issue that's bitten a number of folks is a linker fix that now warns of duplicate symbols which were previously ignored, with one or the other symbol being randomly selected for use. That isn't a problem if both symbols result in the same code being executed, but it leaves open the possibility of some nasty misbehavior and/or a GPF.

Among other things, the linker errors show up on libraries that contain the Windows API. If you've created a LIB file containing a Windows API call, and SoftVelocity's libraries contain that same call, you'll get a duplicate symbol warning. C7 added a bunch of API calls to its win32ext.lib, increasing that risk. One solution, favored by Charles Edmonds, is to strictly use dynamic calls to DLLs rather than linking in the LIB files. Phil Carroll has suggested that the solution is for SoftVelocity to supply a complete LIB covering all API calls.

Is 7.1 "it"?

I can't speak for all Clarion developers, but from my limited use so far, 7.1 isn't the release I've been waiting for. It's a heck of a lot closer, and it may not take much to make it a viable replacement for 6.3. The biggest problem I'm seeing right now is disk writing that borders on disk thrashing - it adds a delay of up to several seconds on some very routine operations such as generating code or existing a procedure, and all those seconds start to add up and make the IDE feel sluggish. I'm not the only one having that problem, but I can't say with certainty that it affects everyone. Perhaps there's some configuration setting that would fix the problem.

Add in the various other bugs being reported, and I think even without the delays caused by writing to disk 7.1 is at least a patch or two away from real usability.

Despite the bugs and a some ongoing issues, Clarion 7.1 is a significant improvement over 7.0, as the above list of fixes and features illustrates. The window previewer is a welcome addition, as is the new report writer. The ability to combine apps into a solution, to have multiple apps open at the same time, better dictionary editing, the use of a real source code editor, color coding in embeds, code completion ... well, let's just say that the overall experience of working in the new IDE (bugs aside) is so far beyond the 16 bit IDE that going back to 6.3 seems like an act of cruelty.

We've all gotten used to waiting for C7 (well, many of us have). SoftVelocity continues to make steady progress. It's not nearly as fast as most of us would like, and the number and severity of bugs in each release is, frankly, disappointing, but progress is being made and certainly that's something to applaud.

In my youth, I did a fair bit of mountain hiking with my friends, and getting to the summit was always a thrill. But along the way there were often false peaks. We'd see what looked like the top of the mountain up ahead, and we'd push on, already weary from the climb, only to find that we'd come to a bump in the ridge, and the real peak was still on ahead.

The analogy isn't perfect, because software, unlike a mountain climb, is never done. But there is a peak in sight, and it's the product release that tips the balance once and for all in favor of Clarion 7. It's close, I'm sure it is. At least I think I'm sure it is. I just can't tell you exactly when we'll reach it.

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, December 31, 2009 by John Hickey

Good article, you definitely hit all the pros and cons. In my case, I've decided C7.1 IS ready (enough) for prime time and I have moved my major application from C6.3 to C7 (using C6 mode, still waiting for several third-party products to recompile).

Yes there were definitely some trials and tribulations as I moved the 22 apps over. But once I got over the rough spots, I must say C7.1 for me has been a huge leap forward.   I have had several very productive days of work using C7.1, and I cannot even imagine going back to C6.3!

Sure it's a bit sluggish at times and code indenting seems to have a mind of its own, but I expect that to improve over time. For now, I am quite pleased with 7.1 and I'll admit I do smile a bit every time I run it :)

Posted on Friday, January 01, 2010 by Dave Harms

Thanks John - I'm glad to hear you're able to move to 7.1. This really needs to become the reality for the majority of Clarion developers.

Dave

Posted on Sunday, January 03, 2010 by Chua SP

Hi Dav, a master is a master, a very good write-up indeed!

The frustrution are, why they (softvelocity) force a "SICK C7.0" to call it "A GOLD" at the first place? It's a "TAX collection" tactic by ending "old core subscription" and get Clarion community to pay..... and still keep them waiting and waiting and waiting again!??

Posted on Monday, January 04, 2010 by Dave Harms

Whatever the reason for SV's choice, the lesson is clear: If you're buying a CSP, don't think of it as giving you access to C8; just think of it as giving you access to all the C7 updates. And the same for your next C8 CSP. I can't think of any major release of Clarion that was good to go the very first time.

Or think of the current CSP as giving you access to the C8 betas, but not the final.

Dave

Add a comment

Clarion Magazine

Clarion: A Decade In Review

by Dave Harms

Published 2009-12-31

I've noticed a lot of "decade in review" articles in the media lately, and I thought it might be interesting to have a look at what's happened in the Clarion world in the last ten years.

Coincidentally, the first decade of the new millennium is also the SoftVelocity decade. By my count, SoftVelocity is the third legal entity to own Clarion. Bruce Barrington created Clarion and founded Clarion Software, releasing Clarion 1.0 in 1986. In 1992, JPI and Clarion Software merged to form TopSpeed Corporation. (By Russ Eggen's count, however, there have been four - he recalls buying Clarion in 1986 from Barrington Micro Systems.) And in May of 2000, nine and a half years ago, TopSpeed's VP of Product Development, Robert Zaunere, formed SoftVelocity Inc. and headed up a group of investors who purchased the Clarion product line.

According to A Brief History of Clarion, there have been at least 23 conferences this decade, including:

- Eight in Australia
- Eleven in South America
- Three in Tennessee
- One in Florida (2004)

There were other gatherings, such as those put on by Richard Rose and the UK user group.

Most recently, we've also seen the advent of webinars, thanks to Arnold Young and John Hickey at ClarionLive! In 2009 Arnold and John produced an amazing 38 educational webinars, plus another six user group webinars.

Major product releases include:

- Clarion 5.5, in August 2000
- Clarion 6.0 in Nov 2003
- Clarion 6.1 in May 2004
- Clarion 6.2 in May 2005
- Clarion 6.3 in January 2006
- Clarion 7.0 hand coder's release February 2007
- Clarion# hand coder's release Nov 2007
- Clarion 7.0 first AppGen third party release Oct 2008
- Clarion 7.0 first AppGen CSP release Dec 2008
- Clarion 7.0 gold release April 2009
- Clarion 7.1 release December 2009

The significant technological changes to Clarion during the past decade include:

- The addition of INTERFACEs to the language
- Full support for OS threading
- Theme support
- Legacy template improvements (using ABC class integration)
- RTF improvements
- EIP improvements
- Theme support
- Internet Protocol driver
- In-Memory driver
- Dynamic file driver
- Enhanced SQL support
- Email support
- A completely rewritten IDE
- A rewrite of the Clarion language for .NET (Clarion#)
- Unicode support (C7)
- ClearType font smoothing support (C7)

The Clarion IDE has seen a variety of improvements and bug fixes since the 5.0 release, but for the most part I think it's safe to say that SoftVelocity's IDE energies have been directed toward the new IDE rather than fixing up the old IDE, which has long needed a major refurb or replacement for two reasons.

One reason is the IDE's 16 bitness; since 64 bit versions of Windows don't run 16 bit applications natively, Clarion developers either need to use a 32 bit version of Windows or use a virtual machine or XP mode to run the old IDE.

The other reason for the rewrite is that the old IDE is embarrassingly primitive by modern standards.

Embarrassment aside, some developers have asked why SoftVelocity didn't simply create a 32 bit version of the old IDE long ago. I remember Richard Chapman showing me a 32 bit compile of the IDE at the 1999 DevCon. I didn't see a running IDE; I just saw the compiler chewing through a bunch of source files spitting out 32 bit object code. I'm not aware that the complete IDE ever ran in its 32 bit form. From conversations I've had over the years with Clarion/ TopSpeed/SoftVelocity folk, my understanding is there's some pretty hairy code in the 16 bit IDE that doesn't lend itself to a 32 bit port. Make of that what you will - I suspect it means that no one really understands how the code in question works.

Rewrites of major products, as Bruce Johnson constantly reminds me, seldom go well. The rewrite of CPD to CDD in the early '90s was a disaster that nearly killed the company. Clarion was more or less saved by the brilliance of the JPI team in London, who came up with the Windows version of Clarion, a case of a rewrite that seemingly went much better than average.

Until, of course, those same developers tried to port the Windows IDE to 32 bits. Ah well, you can't have it all.

If the old IDE was complex enough to make a port prohibitively difficult, any new IDE would be much larger and much more complex. Rather than write such an IDE from scratch, SoftVelocity licensed the code to SharpDevelop, an open source competitor to Visual Studio. SharpDevelop is primarily an IDE for .NET development, which fit in nicely with SoftVelocity's plans to create Clarion#, a full-fledged .NET version of the Clarion language.

The bad news

The IDE rewrite hasn't gone anywhere nearly as quickly as SoftVelocity expected. The initial C7 roadmap blog post in August 2006 called for a first release (codename "Clarion Spirit") that year, along with a release of the .NET product (codename "Clarion Hidalgo"). Both were to be full-fledged products with code generation capabilities and templates.

Three and a half years later C7.1 is arguably still a beta product and Clarion.NET is still a hand coding-only release (with an AppGen announced for 2010).

Clarion 7.1 addresses many of the shortcomings of the 7.0 release, and at least a few developers feel it is a full replacement for C6.3. But issues remain, and new features have been added that undoubtedly will need further refinement. Can you finally move all your apps to C7.1 and reap the benefits of the new IDE? For too many developers, the answer is still "not yet."

That's a bitter pill for all of us who have been hoping that 7.1 would be, effectively, the real "gold" release that puts all the concerns over 7.0 to rest. How close is 7.1? Possibly very close. Perhaps only a patch or two away. Perhaps.

Most importantly, even if all of the bugs in 7.1 were fixed tomorrow that wouldn't make for a painless upgrade from 6.x. That's because there are bugs in pre-7 releases of Clarion that all of us have gotten used to, but which are still bugs that needed to be fixed. They include:

- Template parser allowed incomplete template statements
- Control templates remain after control deletion
- Duplicate symbols allowed by linker

These bugs are coming to light *because* of higher quality code in 7.1; quite possibly there are other issues that will arise/have arisen.

So if you're hoping someday for a painless switch from 6.x to 7.x, think again. I'm not saying the pain will be unbearable; it probably won't be all that bad at all, once you get through it. It's just that any pain, after such a long wait, seems like an insult.

Of course, this is software we're talking about, and major upgrades almost always involve some degree of pain.

The good news

Between bugs in the new and old IDEs, and the long wait for C7, it seems like the decade has closed out on a sour note. Certainly there's been a lot of disappointment in the Clarion community, although you could argue the economic difficulties of the past couple of years have contributed some of the negativity.

It's all too easy to get wrapped up in the things that aren't working right and forget how far the product has come. Clarion 7.1 *is* a significant improvement over 7.0, and 7.0 is a *massive* upgrade from the old IDE.

Even if you don't see yourself using 7.x in the immediate future, chances are you will somewhere down the road. Critical bugs aside, in this day and age it makes about as much sense to use the 16 bit IDE as it does to use Windows 3.1 as your development platform.

In 7.1 SoftVelocity has addressed most of the critical deficiencies of the previous release. There is a new window previewer, and there's a beta release of the new report writer. SoftVelocity has also indicated they're open to revisiting owner-drawn menus.

And let's not forget that as we enter the tens (or is that teens?) Clarion and SoftVelocity are both still here, and the product continues to be improved. It's not easy being an independent Windows development tool vendor in an age when Microsoft dominates, and Visual Studio has grown beyond an IDE to become an entire development ecosystem. Yet SoftVelocity has remained viable. That's a good thing.

While 7.1 doesn't appear to be dazzling Clarion developers the way many of us had hoped, it is another step forward and hopefully just a patch or three away from being usable by the majority.

The future of Clarion

The future of Clarion Win32 development seems pretty clear: it's Clarion 7.x At some point, barring unforeseen disaster, we will have a version of Clarion 7 that is good enough for the vast majority of Clarion developers to use. It won't be a painless upgrade, but it will settle everyone on a new IDE that should suffice for quite a few years.

If you're doing .NET development, and looking to Clarion.NET and the Clarion# language, the future is a little muddier. A .NET AppGen has been announced for early 2010, but we really don't know what that product will look like. For now Clarion.NET is still a hand coder's product.

Many questions remain unanswered on the .NET front. When will there be a usable AppGen? The new template language is based on Microsoft's T4, but what will the implementation really look like? How will Clarion.NET support Windows Presentation Foundation and Silverlight, the current state of the art in .NET user interface development? What are the pros and cons of Clarion.NET having its own standalone IDE rather than being a VS plugin? How does the Clarion# language compare to C# and VB.NET? Does Clarion# have a role to play? Can we reasonably expect an upgrade path from Win32 to .NET? How different is .NET development from Win32 development anyway?

Join me next month as I dust off the ClarionMag crystal ball, peer into Clarion's future, and try to answer these questions.

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 Monday, January 04, 2010 by Tom Drum

I'm actually pleasantly surprised by the improvements to 7.1. I haven't converted any 6.3 apps yet, but this is mainly due to 3rd party add-ons that I can't purchase the upgrades for yet (money constraints). But, I have created several small apps in 7.1 and have not found any drawbacks. I LOVE the new IDE! Like any new product, it just takes time to learn the layout, but it's a quick learning curve. I truly want to thank SoftVelocity for listening to their users' complaints, concerns and suggestions. For a small software development company, I think that they have been and are continuing to bust their butts to make 7.X a viable, reliable product to develop with until the day that a .NET product is ready. Hopefully I won't be "six feet under" by then. :-)

Add a comment