Clarion Magazine

This edition includes all articles, news items and blog posts from March 1 2011 to March 31 2011.

Clarion News

Read 10 Clarion news items.

Articles

Highlighting Text With RTF The Easy Way, Part 2

March 1 2011

Using test-driven development techniques and the ClarionTest framework, Dave Harms concludes his series on displaying color highlighted text by showing how to test and build the CM_Text_RTF class.

Tip of the Week: The Class Browser

March 8 2011

Clarion's template system includes a utility for browsing all the ABC and ABC-compatible classes available to Clarion.

Tip of the Week: Multi-App Solutions

March 9 2011

Opening multiple apps at once in C7 can be a bit confusing; what's all that about adding projects to solutions? Here's what you need to know, plus an easy way to work with custom sets of apps.

Clarion# Language Comparison Updated

March 15 2011

Mike Hanson has prepared a cross-reference showing Clarion# equivalents to VB.NET and C# statements. Topics include program structure, comments, data types, constants,

enumerations, operators, choices, loops, arrays, functions, strings, exceptions, namespaces, classes, interfaces, construtors, using object, structs, properties, delegates, events, and I/O. Based on a VB.NET/C# document by Frank McCown. Updated March 15 with some new examples including AUTODISPOSE and INLINE.

NetTalk 5 Review, Part 2

March 24 2011

In this second of two parts Clarion Magazine looks at NetTalk's extensive support for web application development.

Tip of the Week: Speeding Up The Search Window

March 25 2011

Is your C7 search window taking too long to load? John Hickey knows how to fix that.

Creating AddIns For The C7 IDE

March 29 2011

The Clarion 7 IDE is built on top of the SharpDevelop IDE, which has a highly pluggable architecture. Brahn Partridge shows how easy it is to plug code into the IDE and creates an addin that searches Clarion Magazine.

Tip of the Week: Two Ways To Look At Your Source

March 31 2011

Are you still viewing source code, the embeditor or embed points with just a single, boring editor window? Do you feel like something is missing from your life?

Converting Between Clarion And Excel Dates/Times

March 31 2011

Clarion and Excel have different ways of representing dates and times. Pierre du Toit shows how to accommodate and exploit those differences.

Unit Testing Webinar Workshop Takes On Dates/Times

March 31 2011

Recently John Hickey and David Harms hosted a webinar workshop on unit testing, using Pierre du Toit's article on Clarion and Excel dates and times as a source for a utility class. John and Dave learned a few things about the process, and hopefully the participants did too.

Clarion News

Strategy Online's Personal Money Manager Featured By Microsoft

Microsoft's Finance on Windows has a feature article on Strategy Online's Personal Money Manager, a Clarion 6.3 app.

Posted March 4 2011 (permanent link)

iQ-XML 2.77

iQ-XML 2.77 for Clarion 7.3 has been released. Changes include: Ability to handle Binary Objects (Base 64 encoding) for parsing and writing - can handle unlimited file sizes; New XML:GetSoapContent function to quickly access any Soap Node without having to use a Queue or FindNextNode or ReadCurrent Data.

Posted March 17 2011 (permanent link)

C6 Firebird Example

Tim Phillips has created a Clarion 6 ABC application that demonstrates a wide assortment of Clarion techniques against a Firebird SQL database. The example is at Steve Parker's download center. Look for Tim Phillips or 2 Mar 2011 or Example of Using SQL on Firebird. This application's code makes use of the ABCFree templates. You will need that template set installed on your PC to open and compile the APP file. The ABCFree templates are available from http://www.authord.com/products/Clarion/.

Posted March 17 2011 (permanent link)

Third Party Opportunity For PDF-XChange Viewer/Editor

The upcoming Version 3 of the PDF-XChange Viewer will open access to the Viewer functionality to a far greater extent than has been the case so far, providing an opportunity for developers to access the PDF-XChange end-user installation base. In depth technical information will be be released later this spring as the API and structure of the Plugin SDK interface are finalized.

Posted March 17 2011 (permanent link)

MAV direct ODBC 1.00

MAV direct ODBC (MAV) 1.00 is now available. With this release both versions (Library

and Template) have been combined into one product. All customers who purchased any MAV version (Library version or Template version) in 2010-Jan 2011 will receive a subscription for the whole year of 2011. All customers who purchased it before 2010 will have to renew their Subscription.

Posted March 17 2011 (permanent link)

Clarion Templates for Android Development Special Offer

To celebrate the 100th edition of Clarion Live, Moby announces a special offer for the template set MobyOne4Android (Beta version). From March 11th until April 4th, get the complete set of templates for Android, REST Web Service, Client and REST WebService Server for 199 EUR.

Posted March 17 2011 (permanent link)

ProPath 2.0

LANSRAD has released version 2.0 of ProPath, a path manager and file deployment tool for Clarion. ProPath ensures your application stores it's data and INI files in UAC approved locations under Windows 7, Vista and Server 2008. ProPath makes it easy to: Manage single or multiple data locations, even on a Networkl Control file paths with CSIDL values, the Windows Registry or External Control Files; Use First Run Technology to let your users confirm or select a data path on startup; Test your program or troubleshoot customer data with the Developer Mode; Automatically manage the Application INI as well as 3rd Party or other INI files; Control the location of CapeSoft's FM2/FM3 UPG.TPS file with just one mouse click; Optionally let your users browse your data folders in Windows Explorer. In addition to these standard features, Version 2.0 of ProPath brings two major enhancements, support for multiple data sets and FirstDeploy technology for the runtime deployment of files.

Posted March 17 2011 (permanent link)

ChartPro Wrapper Template 1.0

Version 1.00 of the ChartPro Wrapper Template for the Codejock ChartPro ActiveX control is now available. This is essentially a graphing control that supports numerous different Chart / Graph types including Area, Bar, Bubble, Candle Stick, Line, Fast Line, Funnel, Pyramid, Gantt, High Low, Pie, Point, Range Bar, Spline Area, Stacked Area, Stacked Bar, Stacked Spline, 100% Stacked Bar, 100% Stacked Area, Side-by-Side Stacked Bar, Scatter Line, Step Line, Stacked Spline Area, Doughnut, 3D Pie, 3D Doughnut, 3D Torus, 3D Pyramid and Rotated Bars. In addition to supporting several popular charting styles, ChartPro also supports zooming and scrolling, secondary axes, multiple diagrams and markup titles. The wrapper template enables you to use the control within your application within minutes. It includes a Global Extension template, a Window Control template and a very feature rich Class. The template already includes various features including the ability to load your data via a pre-defined queue, the ability to save a Chart to an Image File (Jpeg,

Bmp, Gif, Png etc etc) plus lots more. The cost of the wrapper template is $95 if purchased on its own. It is also included in the SuitePro bundle and all users who have a current subscription will now be able to download it from the members area of the web site.

Posted March 29 2011 (permanent link)

First Public Release of Moby1 for Android

The Moby Team is pleased to announce the first public release, Version 1.0 beta of MOBY1 for Android, which is now available for purchase.MOBY is a set of Clarion templates simplifying the creation of native applications for mobile platforms. The programmer defines the database structure: tables, columns and relationships. Then - using the templates - he/she generates source code of applications in the native language (as for Android, the language is Java). In the case of simple applications aimed at data browsing and editing and exchanging them between the device, the server and other mobile devices, the knowledge of a programming language is not required. Currently, with MOBY ONE, in the case when the mobile application has to process the data (typical UI excluded), a modification of the generated code will be required - involving the knowledge of the programming language of the target platform. The planned MOBY2 will enable the specification of data processing within MOBY using the Clarion programming environment. MOBY is currently under development. Templates allowing to generate an Android application and generating REST Web Service code are in their beta stages and are available. Templates generating code for iPhone/iPad and for devices running Action Script/Flex (like Blackberry playbook) are under development. The Standard Version generates Android application code which enables to browse and edit data in the smartphone. The functionality of this application is similar to the Clarion generated application (except report procedures). The Enhanced Version generates an Android application like the Standard Version and has an additional option of exchanging data with the external Server SQL, using the service REST. A server side WEB service generator template is also available. It generate the C# code for the REST Service application and the SQL script, which allows you to create a complete database (tables, relations, stored procedures, views) for the Web Service. Using the Enhanced version and REST Service you can synchronize data between mobile application and your Clarion application. Beta prices: Standard Version, 99 EUR; Enhanced Version, 149 EUR; REST Service generator, 149 EUR; Bundle (Enhanced + REST Service) 249 EUR.

Posted March 29 2011 (permanent link)

Brazilian DevCon 2011

The twelfth DevCon Clarion Rio will be held from June 23-26, 2011.

Posted March 29 2011 (permanent link)

The ClarionMag Blog

Highlighting Text With RTF The Easy Way, Part 2

By Dave Harms

Posted March 1 2011

In Part 1 I explained a fairly simple requirement to display text with highlighted words using the RTF control, loosely based on an article by Steve Bottomley. I also indicated I would take a Test-Driven-Development (TDD) approach to writing the needed class, and I set up my test DLL and created an initial test. I also stubbed out a class so my test DLL would actually compile and I could begin running tests.

In this concluding article I'll walk through the process of writing that class in a TDD environment.

RTF document components

As explained in the previous article, there is some standard header text that goes into each RTF document. Although I may want to make this information configurable, for now I'll simply set up a variable in the class to hold the header text. I'll also set up a variable to hold the text I want to display in the RTF control. I'll also need a constructor to initialize the header.

```
CM_Text_RTF                  CLASS,TYPE,MODULE('CM_Text_RTF.clw'),LINK('CM_Text_RTF.
HeaderText                       cstring(500)
Text                             cstring(5000)
Construct                        procedure
GetText                          procedure,STRING
HighlightText                    procedure(string text,long fgColor)
SetText                          procedure(string text)
                                     end
```

Here's the new method:

```
CM_Text_RTF.Construct        procedure
     code
     self.HeaderText = '{{\rtf1\ansi\ansicpg1252\deff0\deflang4105' |
          & '{{\fonttbl{{\f0\fnil\fcharset0 MS Sans Serif;}}'
```

And I can start building my return string:

```
CM_Text_RTF.GetText          procedure !,STRING
     code
     return self.HeaderText
```

I press F8, C7 compiles my solution and loads up ClarionTest and executes the one test currently in the DLL.

Instantly I see that my unit test still fails, and in fact it's not that easy to tell from ClarionTest's display how close I'm getting. So I'm going to add another global include to my test app to help with debugging:

```
include('CM_System_Diagnostics_Debugger.inc'),once
```

And at the end of my test procedure I add this code:

```
     gdbg.write('Expected: ' & ExpectedText)
```

```
gdbg.write('Result  : ' & rtf.GetText())
```

`gdbg` is a default instance of the debugger class, in case I'm too lazy to instantiate one.

The `gdbg.write` statements will write out the expected and the actual text in two successive lines in the Windows debug log, which I can view with the free DebugView utility (Figure 14).

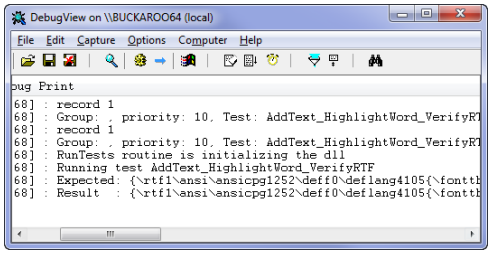

Figure 14. Viewing the gdbg.write statements

By scrolling to the right I can easily see at which point my text varies from the expected text. I find I use DebugView a lot when I'm doing TDD.

I can see from DebugView's display that the header portion of my RTF string is fine; now I need to work on the color table.

The color table

The color table deserves a little bit of thought. First, although I'm passing in one of a limited number of color equates, the actual possible number of colors is 256*256*256, one byte each for red, green and blue. So I probably don't want to limit myself to a subset. What I really need to do is create a color table entry for each color in use, and then keep track of those colors and their position in the color table.

And although I'm only making one call to the `Highlight` method, clearly I could make any number of calls, and each one could use the same color or a different color. So I need to keep track of two things: the list of colors in use, and my highlighted terms and their colors.

The obvious solution is to use a couple of queues.

You can't declare an actual queue inside a class, instead you need to declare a typed queue outside the class and then add a reference to the class.

Here are my typed queues:

```
CM_Text_RTF_ColorsQueue         queue,TYPE
Color                               long
                                       end

CM_Test_RTF_HighlightsQueue     queue,TYPE
Texst                                cstring(100)
FgColor                             LONG
BgColor                             LONG

                                       end
```

Note that I'm starting to think ahead and I've added separate foreground and background color variables, although for now I'll just use the foreground color.

Here are the queue references in the class:

```
ColorsQ                             &CM_Text_RTF_ColorsQueue
HighlightsQ                         &CM_Test_RTF_HighlightsQueue
```

These queues need to be created when the class is instantiated and destroyed when the class is destroyed. I already have a constructor, so I only need to add a destructor method:

```
Destruct                            procedure
```

And here is the code for the constructor and destructor:

```
CM_Text_RTF.Construct           procedure
     code
     self.HeaderText = '{{\rtf1\ansi\ansicpg1252\deff0\deflang4105' |
          & '{{\fonttbl{{\f0\fnil\fcharset0 MS Sans Serif;}}'
     self.ColorsQ &= new CM_Text_RTF_ColorsQueue
     self.HighlightsQ &= new CM_Test_RTF_HighlightsQueue

CM_Text_RTF.Destruct           procedure
     code
     free(self.ColorsQ)
     dispose(self.ColorsQ)
     free(self.HighlightsQ)
     dispose(self.HighlightsQ)
```

I press F8, and everything compiles. Because I've set up the post-build task, ClarionTest pops up instantly and reminds me that my test still fails. There's more to do! I'll need to store my highlighting information in these queues. Here's the new `Highlight` method:

```
CM_Text_RTF.HighlightText       procedure(string text,long fgColor)
     CODE
     self.ColorsQ.Color = fgColor
     get(self.ColorsQ,self.ColorsQ.Color)
     if errorcode()
          self.ColorsQ.Color = fgColor
          add(self.ColorsQ,self.ColorsQ.Color)
     END
     self.HighlightsQ.text = text
     get(self.HighlightsQ,self.HighlightsQ.text)
     if errorcode()
          self.HighlightsQ.text = text
          self.HighlightsQ.FgColor = fgColor
          add(self.HighlightsQ,self.HighlightsQ.text)
     END
```

For each color I make sure I have a matching record in the colors queue; I also make sure I have a matching record for each bit of text I want to highlight.

I almost forgot - I need some code in my `SetText()` method:

```
CM_Text_RTF.SetText             procedure(string text)
     code
     self.Text = text
```

I'm using an arbitrary 5000 character cstring for `self.Text`. This is fine for my current needs, but in the future I may want to change it to a string reference and create a new string dynamically.

At this point I should have enough data to actually build my RTF string. And that's the job of the `GetText()` method. There's a lot of string handling happening in this method, and I don't really like reinventing the wheel so I'm going to use the `CM_System_String` class a lot. This class is a slightly modified version of Rick Martin's StringClass.

```
CM_Text_RTF.GetText             procedure !,STRING
rtfStr                              CM_System_String
documentStr                         CM_System_String
x                                   long
     code
     rtfStr.Assign(self.HeaderText)
     rtfStr.Append('{{\colortbl ;')
```

```
    loop x = 1 to records(self.ColorsQ)
         get(self.ColorsQ,x)
         rtfStr.Append(self.GetColorTableEntry(self.ColorsQ.Color))
    END
    return rtfStr.Get()
```

The `GetColorTableEntry` method is very simple:

```
CM_Text_RTF.GetColorTableEntry  procedure(long color)!,string,private
clr                                 Long
rgb                                 Group,Over(clr)
red                                     Byte
green                                   Byte
blue                                    Byte
reserved                                Byte
                                            End
    CODE
    clr = color
    return '\red' & rgb.red & '\green' & rgb.green & '\blue' & rgb.blue & ';'
```

The `rgb` group structure provides access to the individual color bytes in a Clarion color (as Larry Sand pointed out in Nifty Window Tricks And Smart DLL Loading).

Figure 15 shows my progress via DebugView in three test iterations. I'm creeping toward the full RTF string, although I'm showing a difference in the color table. My example includes a black color table entry in index 1.

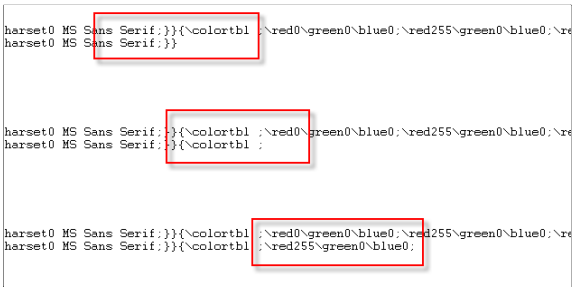

Figure 15. Building the string

A single call to

```
rtfStr.Append(self.GetColorTableEntry(color:black))
```

before the loop takes care of the missing entry.

There's a final color table entry (a very light red) in my expected text which I think can be deleted:

```
\red8\green0\blue0;
```

As I said earlier, I probably don't need the Generator keyword but I'll leave it in, and I'll add the remaining header text:

```
    rtfStr.Append('}{{\*\generator Msftedit 5.41.21.2509;}')
    rtfStr.Append('\viewkind4\uc1\pard\cf1\highlight0\f0\fs17 ')
```

I press F8 to compile the source and run the test. It still fails, but I can see via DebugView that everything now matches all the way up until the beginning of the document text.

Search and replace

Finally I'm ready to create the document text. I have the raw text, and I need to surround each highlighted item with the appropriate `\cf` keyword.

The `CM_System_String.Replace` method makes this relatively easy:

```
docStr.Assign(self.text)
loop x = 1 to records(self.HighlightsQ)
     get(self.HighlightsQ,x)
     self.ColorsQ.Color = self.HighlightsQ.FgColor
     get(self.ColorsQ,self.ColorsQ.Color)
     docStr.Replace(self.HighlightsQ.Text,|
          ' \cf' & pointer(self.ColorsQ)+1  & ' '|
           & self.HighlightsQ.Text & ' \cf1 ')
END
rtfStr.Append(docstr.Get())
rtfStr.Append('}')
return rtfStr.Get()
```

This is actually getting pretty close to the real thing, although there may be some issues with extra spaces.

```
fs17 The word \cf2 red \cf1 should be displayed in \cf2 red\cf1  .\cf3}
fs17 The word \cf2 red\cf1  should be displayed in \cf2 red\cf1 .}
```

Figure 16. The document section

At this point I really need to do a test with the RTF control and just eyeball the results. That won't form part of the unit test, but I do need to do it now to verify my control data.

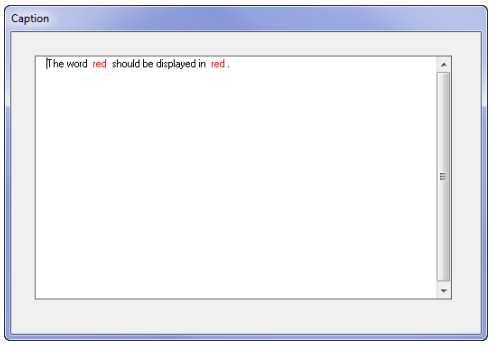

Figure 17. Problems with spaces

It turns out that if I add a blank space after the keyword then everything displays properly:

```
docStr.Replace(self.HighlightsQ.Text,|
     '\cf' & pointer(self.ColorsQ)+1  & ' '|
      & self.HighlightsQ.Text & '\cf1 ')
```

There's one little trick in this code. Note that I'm using `pointer(self.ColorsQ)+1` as the color table index. That code works because the color table is generated in the sorted queue order. The color table is zero based, and `Pointer()` is one-based, but there are *two* initial color table entries generated, one for the default color and one for black. So the first color table entry from the queue is really color table 3, but because the color table is zero index-based the number is really 2 not 3. Which is why I add 1 to the `Pointer()` value to get the correct color table entry. All clear?

Of course, this code isn't perfect, not yet (and probably never). Consider Figure 18.

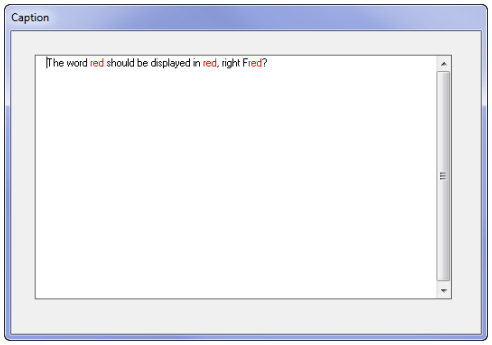

Figure 18. Partial highlighting

My `Replace` method isn't doing a whole-word replace. So I had to add that functionality to `CM_System_String` (that's another story) and add the final parameter to the `Replace` call.

```
docStr.Replace(self.HighlightsQ.Text,|
     '\cf' & pointer(self.ColorsQ)+1  & ' '|
      & self.HighlightsQ.Text & '\cf1 ',,true)
```

Now only whole words are highlighted.

How about multiple highlight colors? Easily done - just add more calls to `Highlight()`.

Figure 19. Multiple highlight colors

The code to create the display in Figure 19 is short and to the point. I created Demo.app with one global include:

```
include('CM_Text_RTF.inc'),once
```

I added an RTFTextControl template to the procedure's window, and pointed it at a local string variable. In the procedure data section I added this code:

```
rtf         CM_Text_RTF
```

And after opening the window, this code:

```
rtf.SetText('The word red should be displayed in red, right Fred?')
rtf.HighlightText('red',color:Red)
rtf.HighlightText('word',color:Blue)
rtf.HighlightText('displayed',color:green)
RtfText = rtf.GetText()
```

With the class in place, that's exactly seven lines of hand code to create the display in Figure 19.

Closing the loop

I've verified the visual appearance of my RTF text, so I now have baseline data I can use for my unit test. If I make further changes to the base class (say, to support boldface or background colors) I no longer need to bring up a window to verify the correct appearance

for that particular test.

You can see the updated expected text value in the `AddText_HighlightWord_VerifyRTF` procedure.

There's still lots to be done on this class. I'd like to add background color and bold support, and probably default font size. You may have other things you'd like to add. If you make changes you think others would find useful, send them to me and I'll see about integrating them into the class.

Summary

I began this RTF class by writing some code that called class methods that didn't exist. That really is the logical place to start. Your class's public methods are the API to your code. If your methods are expressive and understandable, your code is going to be that much easier to use. No one using your class should care how it actually works, as long as it gets the job done.

Testing those public methods ensures that the class works the way it is intended to work. Although my test DLL only has one test procedure, I expect that number to grow as I add more functionality to this class.

In the downloadable source you'll find the test app and the classes, which you may want to move to your libsrc directory. Just look for the files that begin with `CM_`. To run the unit tests you'll need ClarionTest, which you can download from Google Code.

Download the source

Download ClarionTest

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

Article comments

BACK TO TOP

Tip of the Week: The Class Browser

By Dave Harms

Posted March 8 2011

Clarion for Windows grew object-oriented extensions in version 2.0; in version 4.0 the ABC class library made its debut. And somewhere around that time Clarion also acquired an internal class viewer (Figure 1). The class viewer shows you all the ABC-compatible files the IDE can locate on your system (via the redirection file).

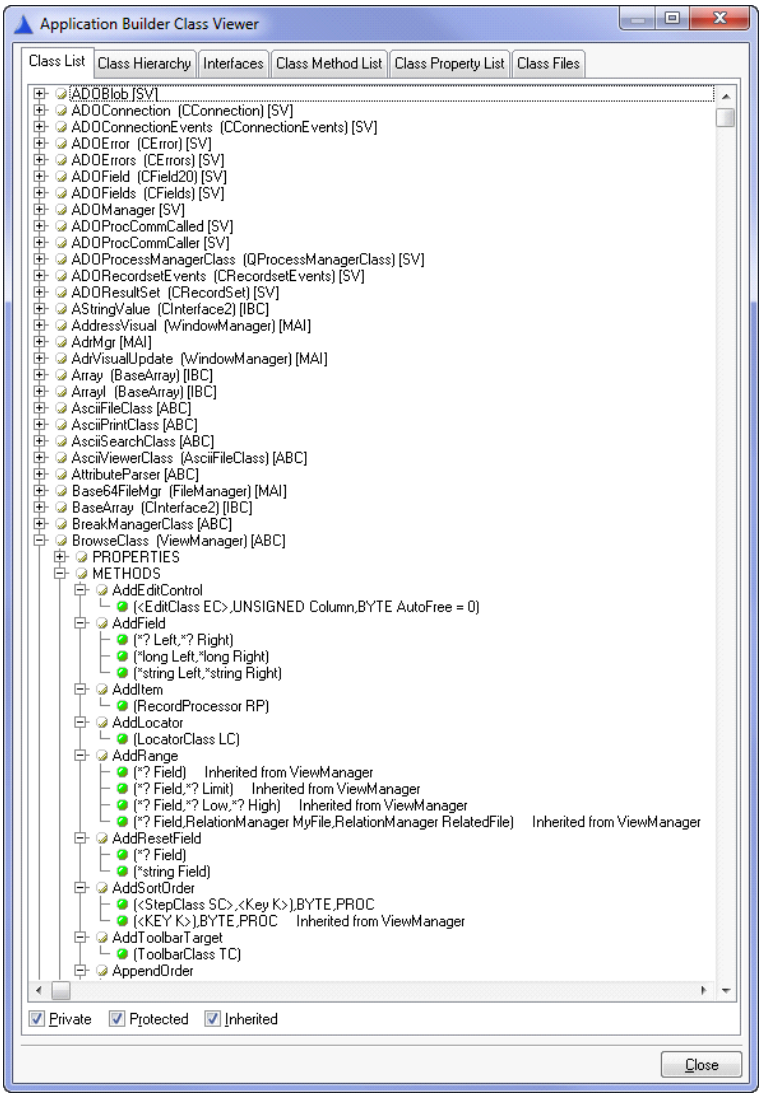

Figure 1. The class viewer

The class viewer is invoked with a template statement, so the only way to bring it up is via a utility template or a prompt somewhere in another template. In either case you need to have an application loaded.

The utility template approach is always handy. From the Application menu choose Utility Template, then select the ViewABCClasses template (Figure 2).

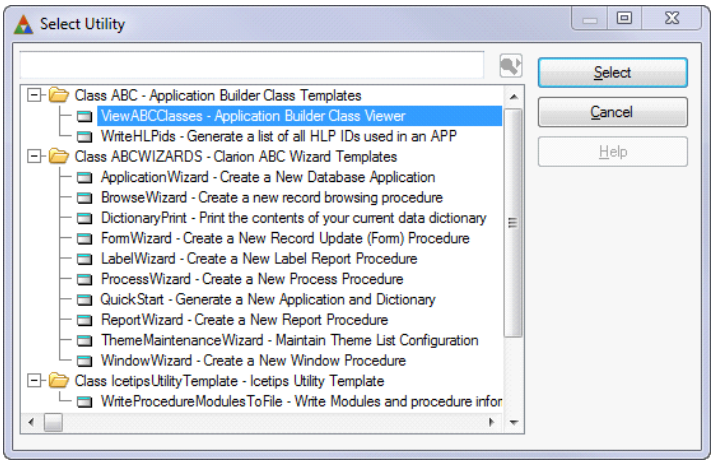

Figure 2. Launching the class viewer

The other way to launch the class viewer is to keep an eye out for the Classes tab in template prompts. You'll typically see a button on that tab that will launch the class viewer. Figure 3 shows the classes tab for a browse control.

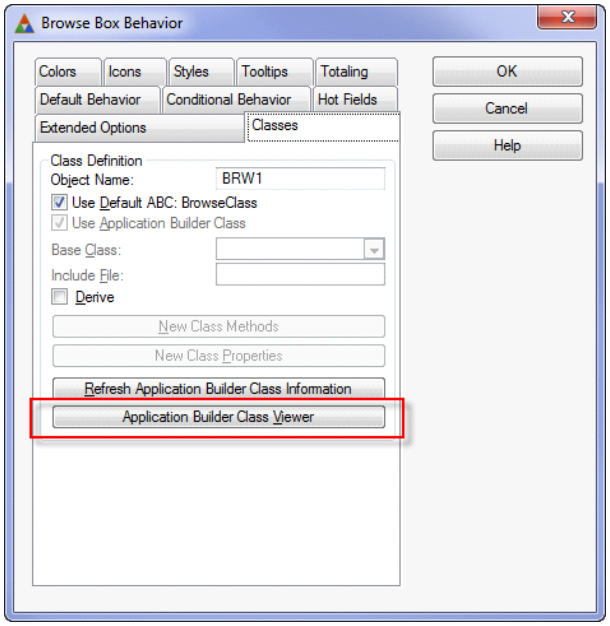

Figure 3. Launching the class viewer from a control template

In either case, what you'll get is a window with a number of different views of all ABC and ABC-compatible classes the IDE can locate on your system (via the redirection file).

There are a variety of different views of the class library, including an alphabetical list of methods, an alphabetical list of properties, class files, interfaces, and a class hierarchy (Figure 4).

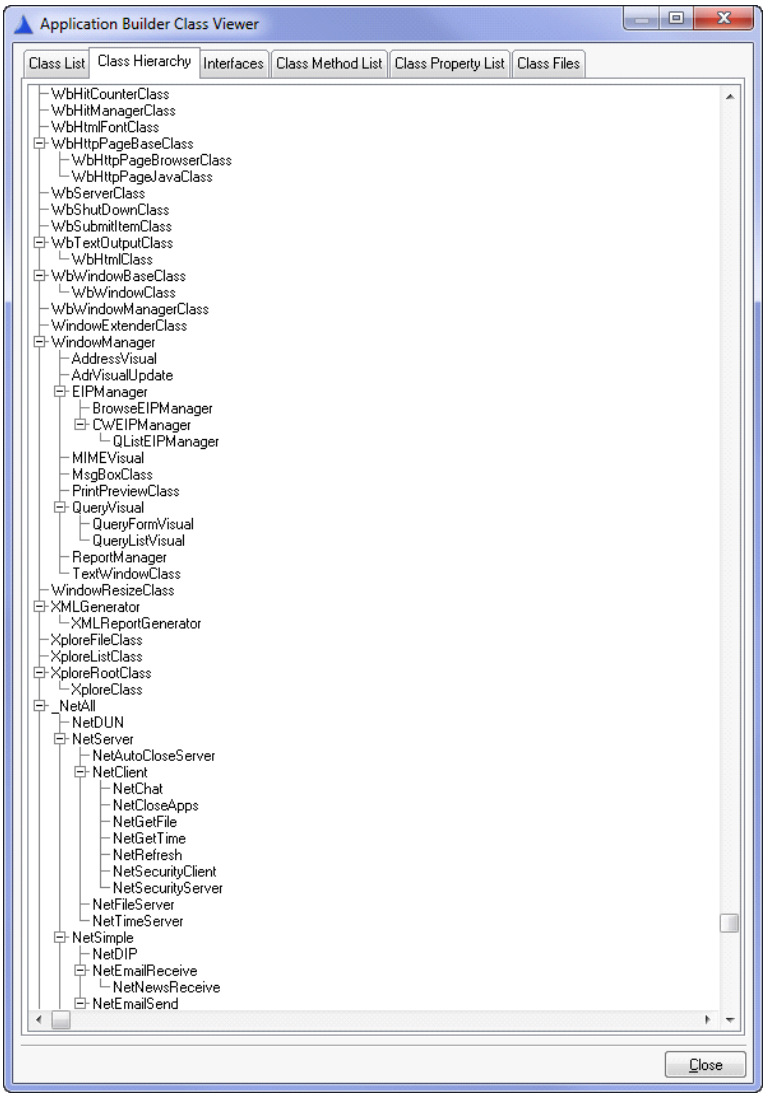

Figure 4. The class hierarchy

Unfortunately, the context menu options are limited to navigation and contraction/expansion - there's no way to, say, double-click on a class and open the source file.

If you want to make the class browser available from your own templates, just use this template statement:

```
#CALL (%ViewABCs(ABC))
```

The %ViewABCs group is defined in ABOOP.TPW as

```
#GROUP(%ViewABCs)
#SERVICE('ClaTPLS.DLL','GenViewABCClasses'),RETAIN
```

This is just a function call into the Clarion template manager that brings up a hard coded window displaying the class library information.

The internal class viewer can be a useful tool, both for exploring the class library and for verifying that the classes you think are installed are actually available to the IDE. If you're looking for a more robust class browsing tool, have a look at Randy Rogers' Clarion Class Viewer.

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

Article comments

by Mark Riffey on March 22 2011 (comment link)

Randy's browser is pretty sweet.

by Dave Harms on March 23 2011 (comment link)

Aye.

BACK TO TOP

Tip of the Week: Multi-App Solutions

By Dave Harms

Posted March 9 2011

Although the Clarion 7 IDE is an amalgamation of Clarion IDE code and the SharpDevelop IDE, there's also a fair bit of Microsoft technology under the hood. In particular, all source and other files needed for compilation are managed using MSBuild, Microsoft's build tool. In MSBuild terminology, a *project* (as defined in a .cwproj file) contains all the files needed to build an application (or a DLL or LIB); a *solution* (as defined in a .sln file) can contain multiple projects.

So where do .apps fit in? If you open a single .app file in the C7 IDE, and there is no pre-existnig project or solution, the IDE will create a solution with the .app's name, and a project with the .app's name. If the .app is called My.app, you'll have at three files (plus any needed dictionary) before you generate or compile:

```
My.sln
My.app
My.cwproj
```

The solution file (My.sln) is the "container" for both the .app and the project. Prior to C7 you didn't need a separate project because all of that information was contained inside the .app file. Actually it still is in C7, but MSBuild has no idea what to do with .app files, so the IDE exports the .app's project data into a cwproj file.

After the first time you open an .APP, you will always have a .cwproj and a .sln for each .app (unless you manually delete these files). And if you open any one of them you'll actually open all three. Usually.

There's an exception to this rule. Let's say you have three apps, A.app, B.app and C.app.

If you open A.app, and you have the app open and you try to open B.app, you'll see the message in Figure 1.

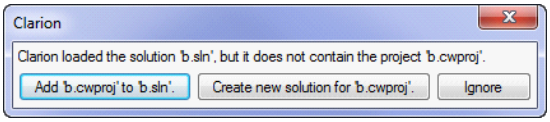

Figure 1. Opening a second .app

In fact this message is incorrect. The solution currently open is A.sln, and what the message should say is "Clarion would like to load B.app and B.cwproj, but they aren't in A.sln. Would you like to add them to A.sln?"

If you want to add B.app to A.sln, click the first button. Otherwise click Ignore.

It can be convenient to have multiple apps in a solution, but usually I don't recommend adding one app to another app's solution. Instead I suggest you create a new empty solution and add as many apps as you like to that solution.

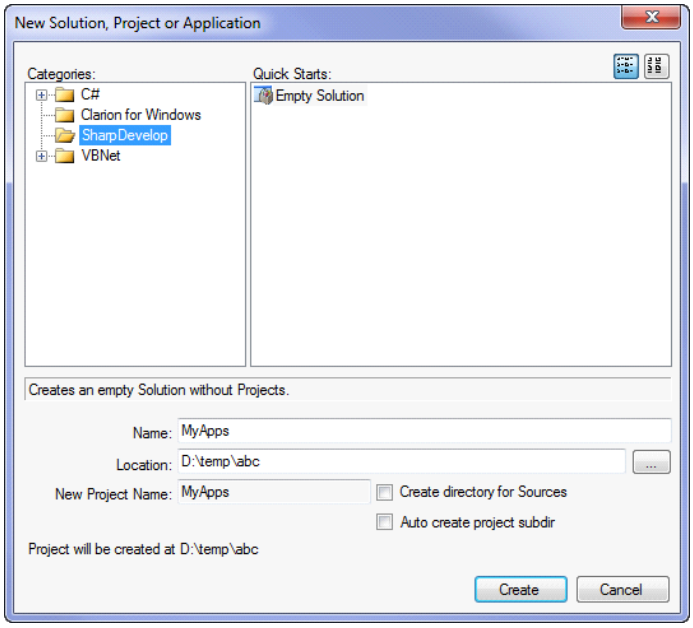

Figure 2. Creating an empty solution

You won't find the empty solution under the Clarion for Windows category; it's in the SharpDevelop category. Doesn't matter: to MSBuild, a solution is a solution is a solution.

Once you've created the solution you can add whatever apps you like (Figure 3).

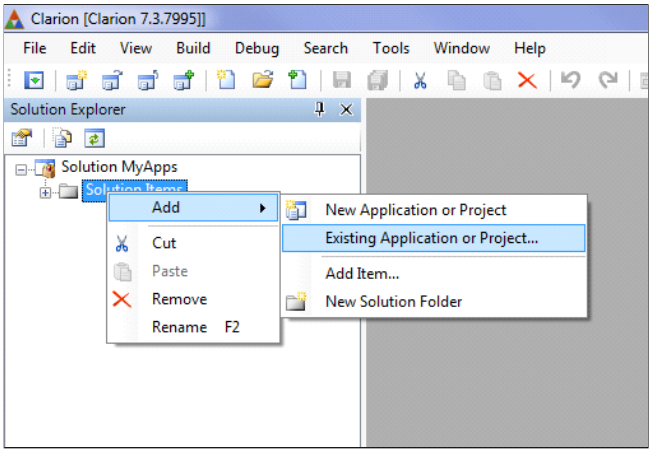

Figure 3. Adding existing apps to the empty solution

Any one app can belong to any number of solutions; if you have a lot of apps you may want to create a number of solutions corresponding to different app-related tasks.

For a more detailed discussion of projects and solutions, read Understanding The C7 Build System).

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

Article comments

by Russell Eggen on March 12 2011 (comment link)

Nice! A bit better than how I created new solutions, especially when the apps did not yet exist (only in TXA form). In this case, your method of creating a new empty solution is ideal. Then use the app pad to create new apps from TXA files.

Another way I create new solutions is not to have any .sln or .cwproj files in my development folder. Open the main app (the EXE) and let it open and create the apps for the DLLs. One of my favorite methods since it goes all the way through without asking (you do need to change the default setting).

Regardless of how I build the solution, I also tend to rename it (press F2). Then if I need anything else, I add them to the solution by hand, like my dct file. Class sources and templates (if you are working on one) may also be part of a solution (but not compiled), makes for a great customized pick list.

---

by Dave Harms on March 14 2011 (comment link)

Thx Russ. Creating apps from TXAs via the app pad? Cool, I'll have to try that!

---

by Robert Wagner on March 15 2011 (comment link)

Great article, and timely. Not having 3rd party things like the Compile Manager still makes me insecure.

But the thing that really perplexes me is the behavior of the IDE, when I go to compile anything after the first app. I compile the first app, everything is fine. But on the second app, I have to click 2, 3, or 4 times on the File menu line to ask the IDE to open an app. This is a general problem. After using the IDE for at least one app, it becomes very unresponsive. Yet, it will eventually respond, if you keep clicking away. In the meantime, god only knows what it is doing, and the chances of hitting the wrong key, or hitting the right key too many times, is very high.

Bringing this up in the newsgroup simply doesn't work - it gets deleted immediately.

Posting a bug report doesn't seem to work either.

I had a dot net developer take a look (maybe that's just the way dot net works...) at my development machine, and his response was that it (the IDE) is just really buggy software. I am forced to agree. If an IDE option is not available, it should be disabled, or hidden. Otherwise the IDE should respond. If it's busy, it should present the hourglass cursor.

---

by Dave Harms on March 15 2011 (comment link)

It's possible some of those delays will go away if you disable write cache buffer flushing for your drive. This problem doesn't affect everyone, or even most, but I've had it crop up twice now, on my personal dev machine and on a shared Windows 2008 server. Of course you need a good UPS to go along with that option - there's some added risk involved.

I think the majority of the IDE's problems stem from the interaction of the AppGen, which is evidently Win32 code, with the rest of the IDE.

BACK TO TOP

NetTalk 5 Review, Part 2

By Dave Harms

Posted March 24 2011

In Part 1 of this review I introduced CapeSoft's NetTalk 5, which is arguably one of the largest and most complex Clarion third party products.

In that article I mentioned that NetTalk's functionality can be divided into two general areas. One is creating Clarion-style business web applications; the other is everything else. In Part 1 I covered the "everything else" aspect (which is generally unchanged from NetTalk 4). In this concluding part I'll take a look at NetTalk's ability to build web applications using Clarion.

Web app development concepts

To understand how NetTalk approaches web app development you need to know a little bit about the differences between web apps and desktop apps.

While Clarion devs often refer to their desktop applications as "multi-user", they really are single-user applications that share data nicely with other instances of the application. That is, only one user can interact with one running instance of the application at one time.

When you load your EXE, it interacts with your computer's hardware (via Windows) to display information to you and to receive input via the mouse and keyboard. The data may be on another computer, but your application is running locally.

It's obvious that you can't have two different users using the same instance of an application at the same time. The app only knows how to interact with your screen on your computer, and with your keyboard and mouse. A second user on another computer is going to have to run a separate instance of the application; no other computer can use your instance.

And yet, this is exactly the kind of trick web servers pull off; they are single executables that can respond to any number of different users at once. They can do this in part because they don't interact with you directly using your keyboard, mouse and monitor. Instead they send you a web page, and when you do some appropriate action in your browser (such as click on a button to submit a form) your browser sends some data back to the server, and the server typically responds with another web page.

So, how do you turn a regular Clarion application into a web application?

You don't. They're just too different.

You can get close by using a thin client solution such as Thin@ (recently reviewed in Clarion Magazine) or ClarioNet. These systems use a custom server and a custom thin client to present just your application's UI to the user. The application itself runs on the server, and there is one instance of the application for every user, just like on a desktop deployment. (This is also how SoftVelocity's InternetConnect and WebBuilder products work.)

Solutions like Thin@ and ClarioNet leverage your existing apps, saving development cost. They do however require your users to install and run the thin client. And not all users may be willing to install that software, or a thin client may not be available for their particular computing platform. Thin client solutions also need more memory on the server, since they run an instance of the app for exe user. And they typically send more data back and forth because they're constantly repainting the thin client's screen.

A true web server, on the other hand, has relatively low memory requirements and, since it's delivering web pages, is often accessible on all popular computing platforms.

The two main web servers are Apache with nearly 60% of sites, and Microsoft's Internet Information Server, or IIS, with 23% of sites. There are many lesser-known web servers, including CapeSoft's very own NetTalk web server. And it's the NetTalk web server that's the key to building web apps with NetTalk.

When you create a web app with the NetTalk templates, you're actually creating a completely self-contained web server. That's an advantage in that you don't need to install or configure IIS, Apache, or any other server. Of course it does mean that you need to have the rights to install and run your app as a web server, but it's not nearly as difficult or expensive to lease a dedicated or virtual server these days where you have complete freedom in what applications you install and run.

Clarion web apps

NetTalk's approach to web applications is to let you continue to use the familiar Clarion AppGen environment, and even the ABC and Legacy template chains. But you don't put together your app using the standard browse/form procedure and control templates; rather you use NetTalk's custom templates.

It takes very little time to create a simple browse/form web app. (I started with an ABC app, although according to the docs you can use Legacy templates as well.)

The first step, after creating an app, is to activate NetTalk via the global extension as shown in Figure 1.

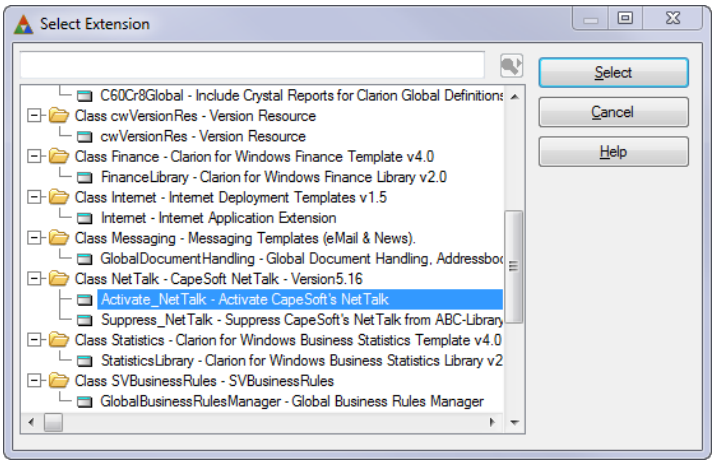

Figure 1. Adding the NetTalk activation global extension.

There are a number of settings on the global template, but most of them are for fairly specific and often quite technical situations. Note the ability to disable NetTalk without removing the extension (and everything that will come to depend on this extension).

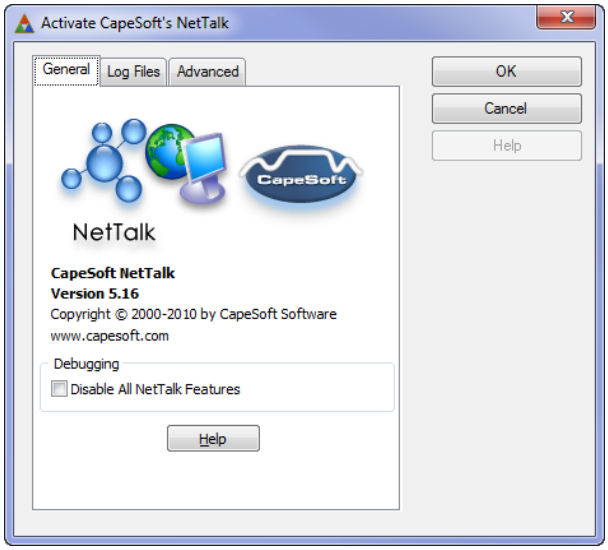

Figure 2. Global extension properties

Once you have this global extension loaded, other templates become available. Still in global extensions, add the NetWebServerGlobal template.

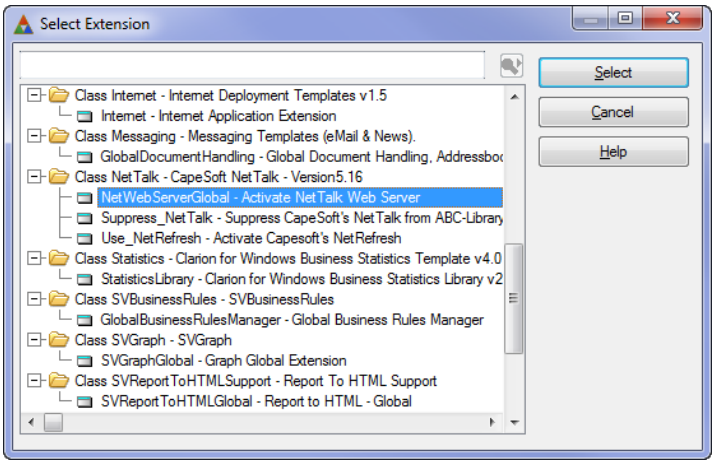

Figure 3. Enabling the web server.

There are a number of options on the web server extension template as well, including support for multi-DLL apps, but none of them need changing at this point.

To actually turn this application into a web server you need to add two procedures. The documentation indicates there are three ways of doing this: import from an application that already has these procedures, create the procedures from scratch using the templates or, the option I tried, run the ImportNetWebServerABC utility template (Figure 4).

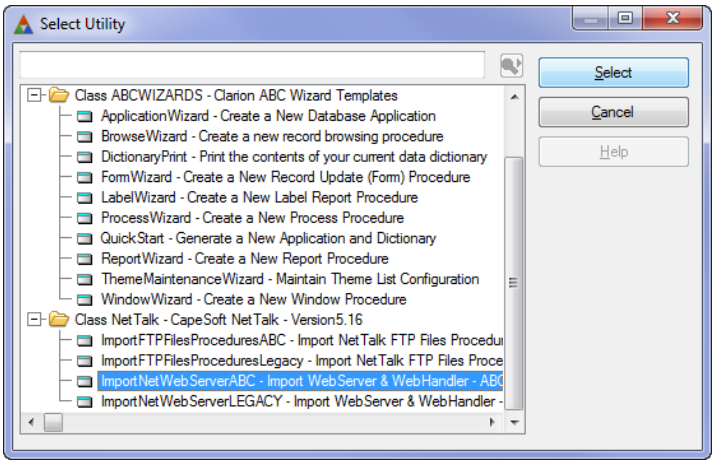

Figure 4. Importing the web server procedures.

The template created two procedures, WebServer and WebHandler.

WebServer is the one that needs to run automatically when the application starts, so I went to Global Properties and changed the First Procedure to WebServer.

Web pages and visual designers

Creating web pages in NetTalk is unlike working with HTML, and unlike working with standard Clarion applications. It's not that you can't do HTML work; you can blend static HTML with the code NetTalk generates on the fly. And you are using the AppGen. But

you're not using the designers. You're doing most things via template prompts.

Not having a window designer isn't as bad is it sounds. In modern web applications the appearance of page elements is determined by the style sheet (the .CSS file), so you really don't need (or even want) to use the Clarion designer for web pages. In fact the visual design aspect is still there, but it's now moved out to the browser. So you use Clarion to create all the components needed for browses, forms etc., and then you run the application and use a browser plugin like FireBug or a runtime CSS editor like Stylizer to make CSS changes and see the result of those changes instantly. And you do create browse and form procedures, but they aren't exactly like a desktop application procedure; think of them as "controls" - blocks of functionality that you insert into a web page using NetTalk's custom tags. So in NetTalk you can actually reuse browse and form controls in multiple pages; this is the equivalent of creating a browse in a desktop application and then using it on more than one procedure, something you just can't do in a standard Clarion desktop app.

Using a style sheet to control visual appearance also means that you can make a style change in one place and have it affect the entire application. In a desktop Clarion application you have to make changes individually to every control on every window, which is tedious and error-prone.

A browse/form example

Getting a web server set up isn't actually that hard, but putting together web page can seem a bit daunting at first. There's a lot of documentation, and it's not always easy to find the bit you need. I skimmed the docs and then dove in; by comparing my code with one of the working examples, I was able to bring up an empty home page. I also managed to add a browse to the page, though at first I couldn't get the browse to display because I'd mistyped the browse procedure name in a tag (I'll explain browses shortly).

After I corrected the tag error the browse came up, but without any formatting and without operative buttons. At the top of the page I saw the text "Error in site JavaScript".

It turned out that I'd neglected to copy over the standard web subdirectory tree, which contains images, scripts and stylesheets. Once I did that my browse displayed normally and I was able to add, change and delete records.

If the documentation is sometimes bewildering (mainly by its sheer volume), there's no faulting CapeSoft for a lack of examples; there are a vast number of these, and you're almost certain to find one that's a good starting point for the kind of web development you want to do. You may find that an easier way to dive in, although I strongly encourage you to read all the documentation through at least once.

Although I was able to create a browse and form pretty quickly (once I got past my obvious-in-hindsight mistakes), rather than walk you through my hack job I'll use Web1.app in the BasicBrowseAndForm (1) directory. If you make and run this application you'll see the

server status window in Figure 5.

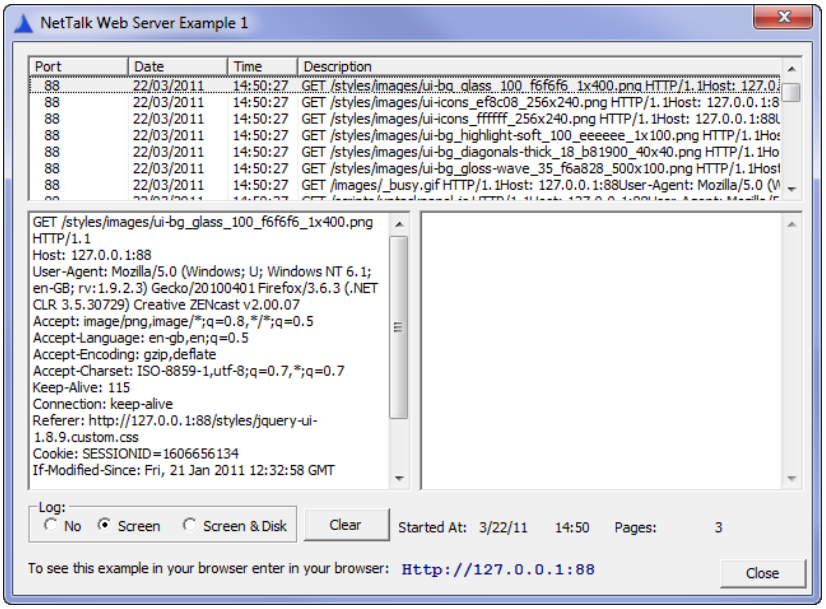

Figure 5. The Web1 server status window

The top listbox on the server status window shows the individual requests made to the server. Any given page typically results in a number of requests, as each image, style sheet, and web page must be requested individually. Click on a request to see the full HTTP request information in the left-hand panel.

The Web1 application uses port 88 and by default runs on localhost, so to browse the site you need to use the url http://127.0.0.1:88. Figure 6 shows the running web application.

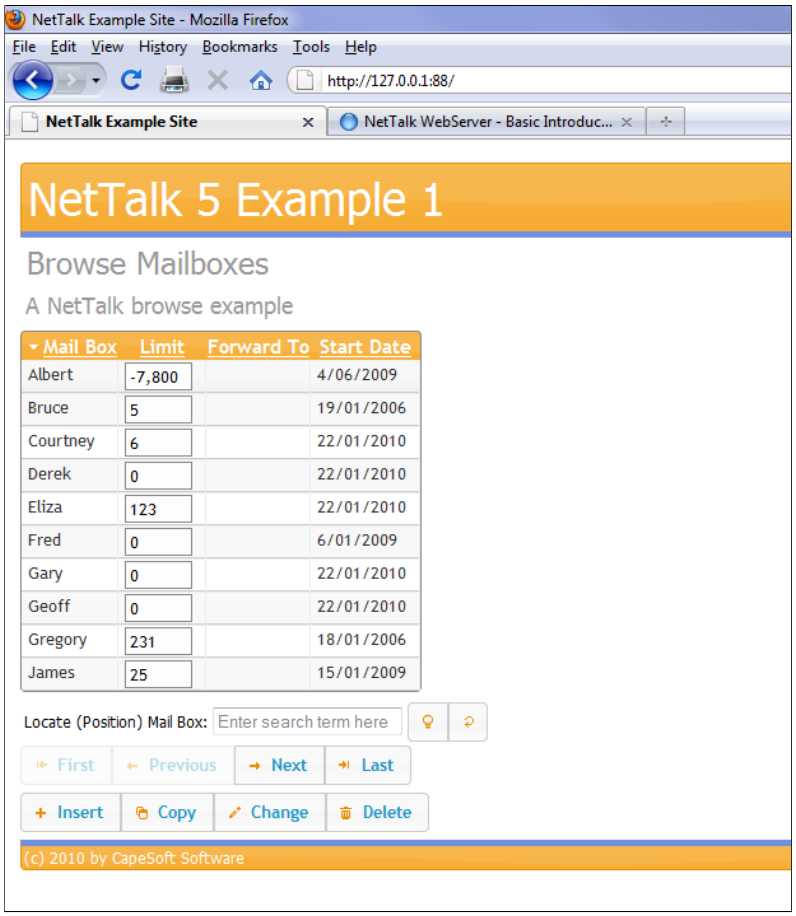

Figure 6. The Web1 applicationb browse

Figure 7 shows the update form, which is a popup window that disables the underlying browse.

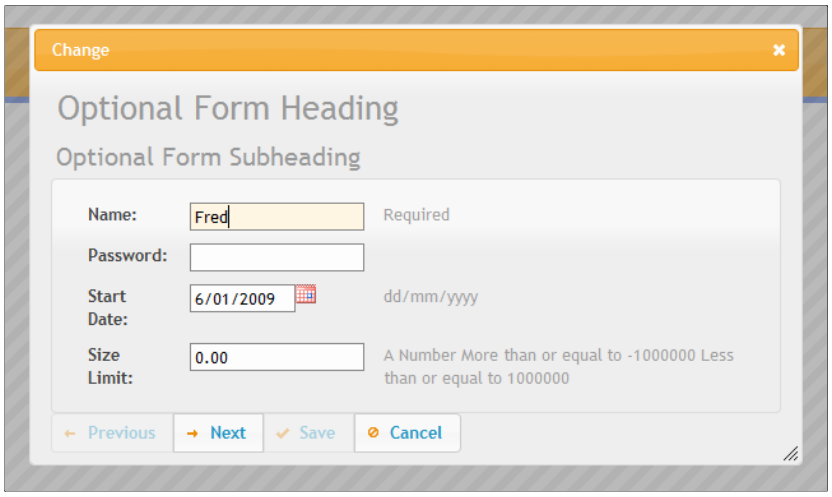

Figure 7. The update form

Figure 8 shows the app tree. Along with the WebServer and WebHandler there is an IndexPage (NetWebPage) procedure. There are also four procedures that are really page

components: one for a browse, one for a form, one for a page header and one for a page footer.

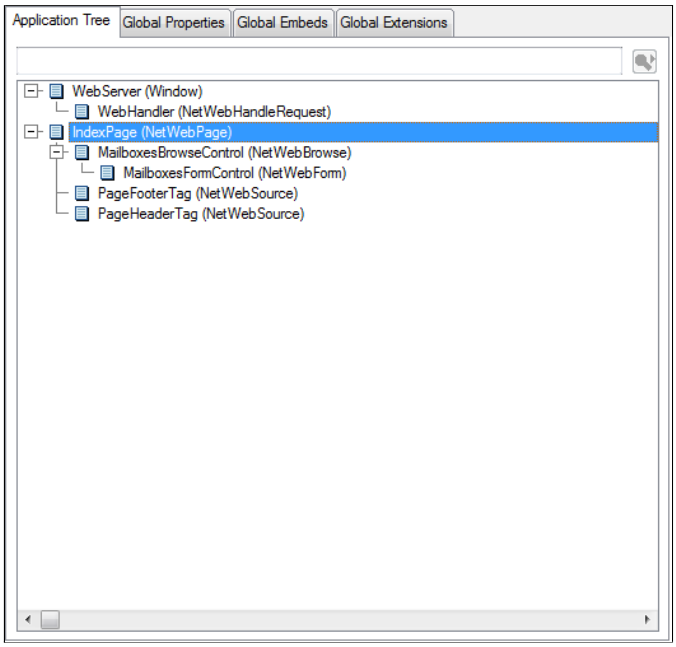

Figure 8. The application tree

Browses and forms are configured via the procedure properties Action button. Figure 9 shows the first tab of the browse's properties.

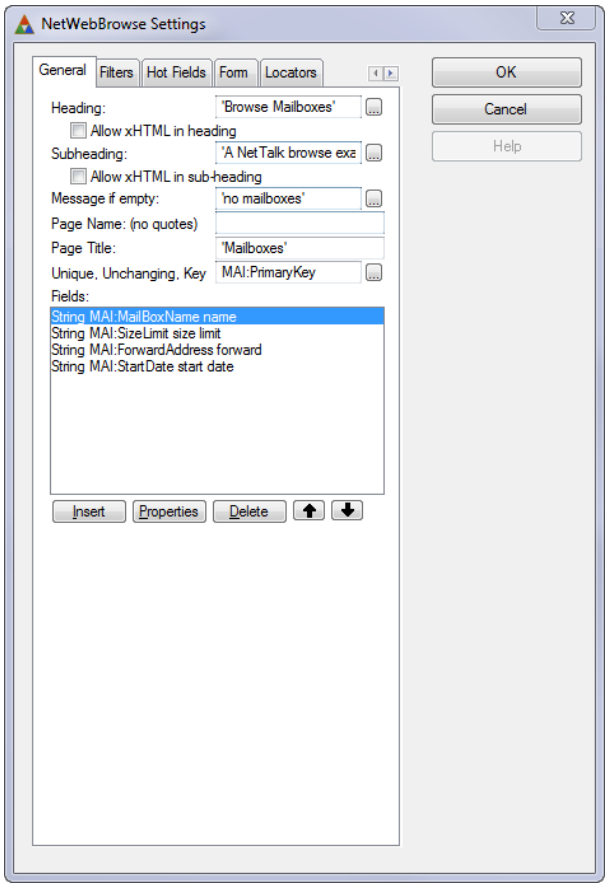

Figure 9. Browse procedure properties

This is also where you specify the form settings (Figure 10).

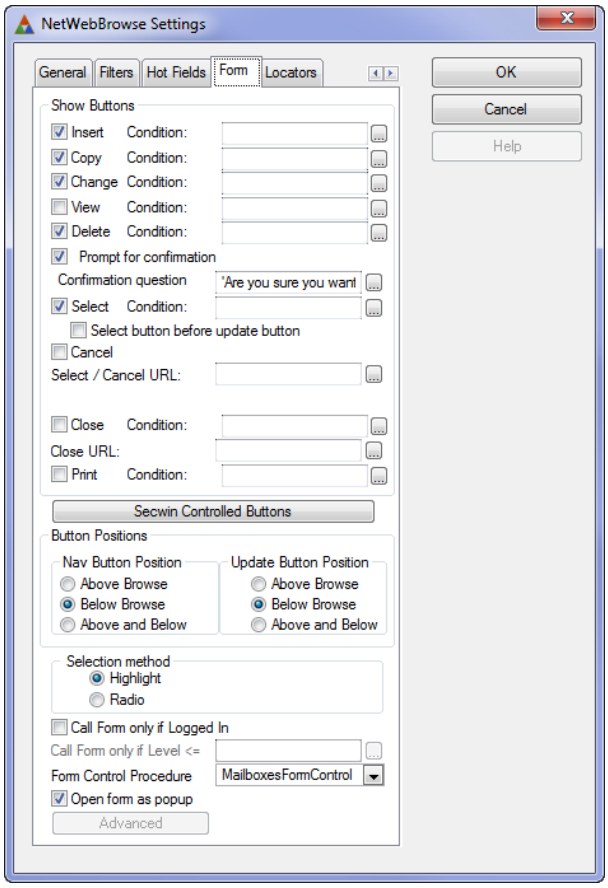

Figure 10. Setting the form properties on the browse

Again, there is no window here; everything that appears on the window is determined by the templates.

Forms are similarly windowless; Figure 11 shows the General tab of the form properties, which bears some similarity to the browse settings.

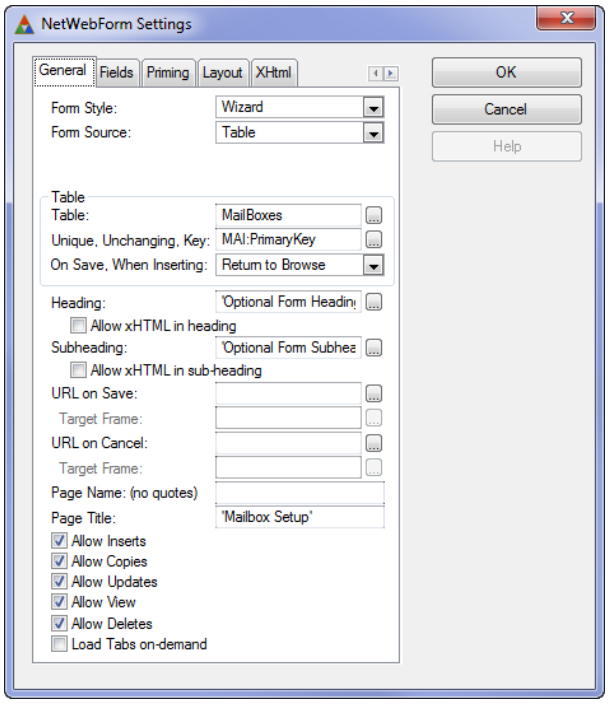

Figure 11. Form settings

The Fields tab (Figure 12) is where you specify the fields to show, and you do so on one or more tabs.

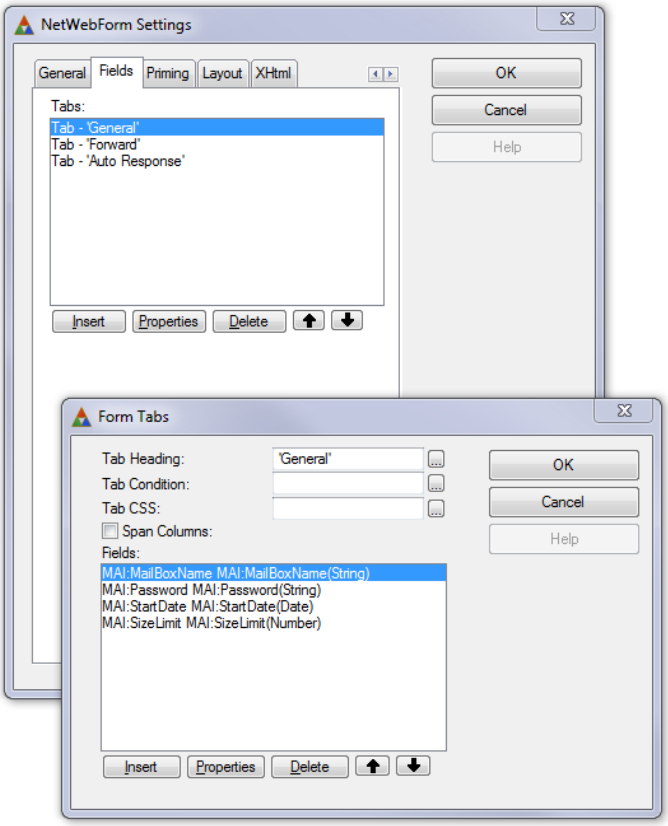

Figure 12. Adding fields to a form

It's easy to see that the form is linked to the browse by a template prompt, but how does the browse get into the web page? Again, it's via the procedure properties Action button, this time on the IndexPage procedure (Figure 13).

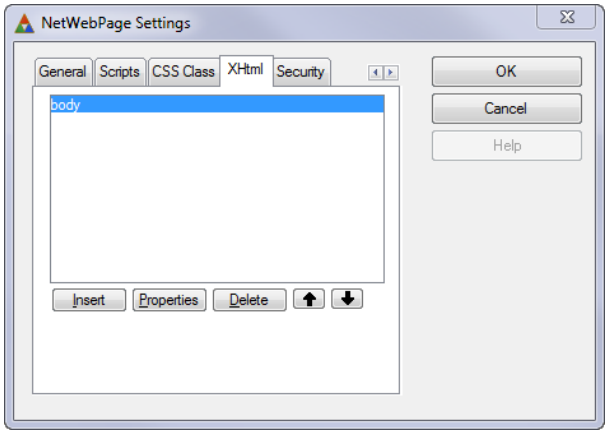

Figure 13. The XHtml tab

The `body` routine properties are shown in Figure 14.

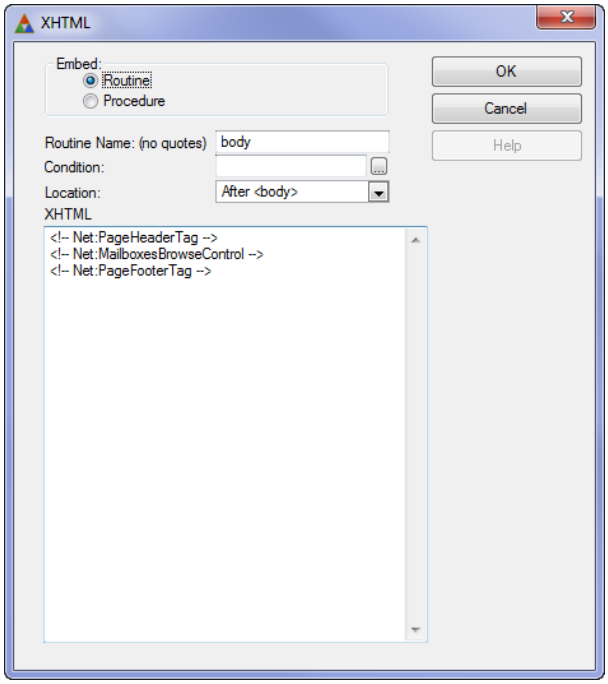

Figure 14. The body routine properties

The XHTML text in Figure 14 includes a number of NetTalk tags. If you look closely you'll see that the text following `Net:` matches the procedure names for the header, browse, and footer procedures. NetTalk substitutes the HTML generated by those procedures in place of the corresponding tags.

Reports

NetTalk supports PDF reports. This is one area where you can reuse existing code. There are a few additional steps to reusing a report procedure in a web app, as detailed in the help. You need to add an extension to the report procedure and you need to change its prototype, but as the parameter is optional you can, if you have your reports in a separate DLL, call the exact same reports from your desktop app as you do from your web app.

JavaScript and jQuery

NetTalk makes heavy use of JavaScript and in particular the jQuery library to enhance the user experience. As of NetTalk 5 the date selector popup is from jQuery, and various other UI components are jQuery-based. NetTalk apps also use ThemeRoller, the jQuery theming toolkit.

The calendar

When I showed a draft of this review to Bruce Johnson, he pointed out that I hadn't said anything about the new calendar control. This is more than just a date picker - it's a full-fledged calendar which you create using the NetWebYear procedure template. Figure 15 shows one month of the calendar view; Figure 16 is a browse view of the calendar data.

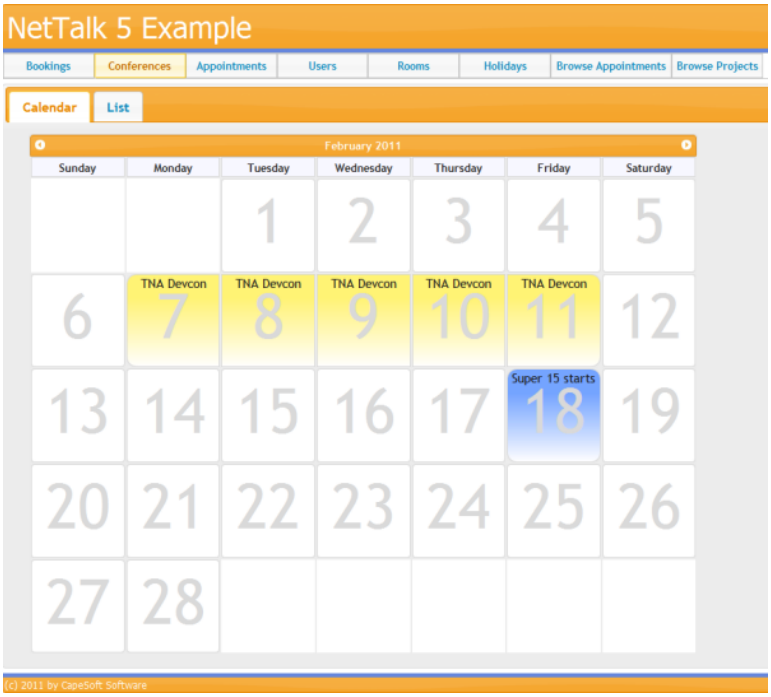

Figure 15. The calendar

Figure 16. Calendar data

Bruce recently did a ClarionLive webinar entirely on the new calendar. And if you're doing web development of any kind you owe it to yourself to watch his webinar on CSS.

Examples

As I've said more than once, NetTalk is an absolutely massive product, and it's not always easy to find the documentation for a particular subject or issue. NetTalk's saving grace is the equally massive number of example apps. Here's a short list of just the web app examples (there are many more examples for the "other half" of NetTalk, as covered in Part 1).

- Browse/form with Windows-style menu
- Browse/form with login
- Framed page with Outlook-style menu
- Single page with browse and form
- SSL
- Login before frame appears
- Server serving both SSL and non-SSL pages
- Server routing all non-SSL pages to SSL
- Graphs (using Insight Graphing)
- Send email from browser
- User access control
- PDF report using C6EE functionality
- PDF report using PDF-Tools
- Filter one browse with selection on another browse
- Multi-DLL

- A browse in an entry form
- Relation update
- Browse on one table, form on another table (may not be working yet)
- Chaining to a second form
- Parent-child browse
- File upload
- Hot fields
- Logging
- Frame with XP Task Panel style menu
- Serving XML
- Add hyperlinks to browses and forms
- Custom error page
- Drop filter
- Calculator
- Time field handling
- HTML editor
- A server built with legacy templates
- Use of CapeSoft Message Box
- Downloading files (HTTP)
- SOAP server
- Security with multiple access levels
- Use of an ActiveX control (IE only)
- Page with progress indicator
- Embedding a browse in a static page
- Tagging
- Locators
- Multi-row browses
- Dictionary and local form field validation
- Exporting to Excel
- Running the server as a service
- Various tab types
- Integration of PHP
- Running multiple sites from one server
- CPCS Report support

As you can see, there really is an example app for almost every situation.

Summary

Developing web apps with NetTalk isn't a drag and drop, visual designer kind of experience.

Because almost everything is controlled by templates, you really don't know what you have until you run the application. But what you get will probably be something much nicer than you could have done anyway if you did have a drag and drop development surface. CapeSoft have done an excellent job of keeping NetTalk up to date, integrating jQuery support and making use of theming and other JavaScript niceties.

The documentation may be a bit scattered at times, but the examples are numerous, the active NetTalk developer community is ready to help, and CapeSoft's support (and code quality) is of a consistently high standard.

Web development gets more important every day; NetTalk is a great way to extend the reach and the life of your Clarion code base.

NetTalk 5 is available from ClarionShop:

- New licence: $599
- Upgrade: $299

Rating: 

Resources

- NetTalk home page
- NetTalk Central - a community for NetTalk users
- The NetTalk e-book ($197)
- ClarionMag's NetTalk review from 2003

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

Article comments

BACK TO TOP

Tip of the Week: Speeding Up The Search Window

By John Hickey

Posted March 25 2011

Recently I noticed that my Clarion 7 Search and Replace window (which I usually bring up with CTRL-F) was taking longer and longer to display. I finally realized that Clarion remembered all of my searches, and that the search history had grown very large! It was taking several seconds for the search window to load all of these items into memory.

I decided to clear these out, and ended up doing some detective work. I finally found that previous search terms are stored in the ClarionProperties.xml file located in %Appdata% \SoftVelocity\Clarion\7.0 . If you are noticing your Search window coming up slower and slower, here is how you solve the problem:

First make sure you exit Clarion 7, then load ClarionProperties.xml into a text editor. There are two XML properties you need to change, `FindPatterns` and `ReplacePatterns`, each of which stores all of the past entries in a single string delimited by ÿ characters. Remove the contents of the value attribute so you end up with these two lines.

```
<FindPatterns value="" />
<ReplacePatterns value="" />
```

Then save the ClarionProperties.xml and run Clarion. Your Search window will be snappy once again!

Article comments

by Robert Wagner on March 30 2011 (comment link)

> This sounds like a bug in C7. Shouldn't the search history be cleared each time the IDE is invoked? At a minimum, shouldn't the IDE have something that allows you to clear the search history?
>
> But, till then, we at least have a work-around.

Thanks John!

---

by Dave Harms on March 30 2011 (comment link)

Robert, I think it's often helpful to have the history there from previous sessions. An option to clear the search history would be good. And really it's not that much data, so perhaps the code that manages the search history is at fault. I imagine C7 simply uses the functionality already present in the SharpDevelop libraries.

---

by John Hickey on March 30 2011 (comment link)

I wonder if Brahn Partridge could make us a nice button to clear search history :)

---

by Bob Roos on March 31 2011 (comment link)

Thank you Detective John.

I was just thinking this was getting slower. Turns out the find had saved 549 items and replace 52. I can see how that could slow the load down because it took me a while to count them all ;) .

Actually the pattern is find-string<195,191>another-find-string<195,191>still-more-find.....

I would propose a user specified limit to the number in the list. I like the history, but rarely use it because whenever I wanted the last entry it was not in the list. If you search for xxx and then search for yyy and come back and click the drop list it is empty. If you close the find window and reopen it then the entries are there. Somewhat worthless feature since it is not there when you may most need it. (A bug most likely).

BACK TO TOP

Creating AddIns For The C7 IDE

By Brahn Partridge

Posted March 29 2011

The new Clarion IDE, as most will know by now, is based on the open source #develop (short for Sharp Develop) IDE v2.1.0.2447. What you may not (yet!) know is that almost all of the "Clarion" parts you see when you open Clarion7 are the result of addins created by SoftVelocity.

The developers of #develop have this to say about the platform structure of the IDE:

> The application itself is based on a small Core System. All functionality is implemented in one or more add-ins which are plugged into the Core. Inside the Core System some basic subsystems ought to be defined which extend the Core functionality

Part of the magic of the #develop system is that addins can be written to extend and enhance almost any component of the IDE, including interacting with other addins. With this article I hope to introduce you to some of the possibilities available by stepping through the creation of an addin that will allow you to search favourite online clarion resource right from inside the Clarion IDE!

At the end of this article will be some links to additional useful articles and resources but for now it is time to dive right in and prepare a development environment suitable for writing addins. I'm going to use the C# language and the free Visual Studio Express IDE (for those .net newbies, don't be put off by the C# code, most of the hard stuff is already handled by the IDE core- an addin can do a lot with very little code). There is no reason not to use the free SharpDevelop IDE, the later versions are quite capable, but I had VS handy.

Setting up your environment

On my machine I've installed Clarion7 to “C:\Clarion7”. Make sure to adjust all paths to suit your setup!

Start by downloading and installing the free Visual Studio Express (remember to get the C# version) (Any IDE or language that can create a .net assembly should work fine, but the examples here are all C#)

Since the source code for the Clarion IDE is not available and there is no way to run it within a debugger, I would also recommend getting to know RedGate Reflector, formerly a free product, now available at a modest cost. Reflector is an assembly inspector and decompiler and is invaluable for learning about how the IDE works.

Lastly, configure the Clarion7 IDE to send DebugOutput information to DebugView, as follows.

The IDE uses something called log4net via a helper class called LoggingService. This is very helpful to see what is going on at runtime. To turn it on, edit `<Clarion install directory>\bin\Clarion.exe.config file` and add this inside the `<log4net>` tag:

```
<appender name="DebugConsoleOutput" type="log4net.Appender.OutputDebugStringAppender">
    <layout type="log4net.Layout.PatternLayout">
        <conversionPattern value="%date [%thread] %-5level- %message%newline" />
    </layout>
</appender>
```

And this goes into the `<root>` tag under `<log4net>`:

```
<appender-ref ref="DebugConsoleOutput" />
```

The complete log4net section of clarion.exe.config file should now look like this (with additions in bold):

```
<log4net>
     <appender name="ColoredConsoleAppender" type="log4net.Appender.ColoredConsoleAppender">
          <mapping>
               <level value="FATAL" />
               <foreColor value="Red, HighIntensity" />
          </mapping>
```

```
            <mapping>
                 <level value="ERROR" />
                 <foreColor value="Red" />
            </mapping>
            <mapping>
                 <level value="WARN" />
                 <foreColor value="Yellow" />
            </mapping>
            <mapping>
                 <level value="INFO" />
                 <foreColor value="White" />
            </mapping>
            <mapping>
                 <level value="DEBUG" />
                 <foreColor value="Green" />
            </mapping>
            <layout type="log4net.Layout.PatternLayout">
                 <conversionPattern value="%date [%thread] %-5level- %message%newline" />
            </layout>
       </appender>

       <appender name="FileAppender" type="log4net.Appender.FileAppender">
            <file value="SharpDevelopLog.txt" />
            <layout type="log4net.Layout.PatternLayout">
                 <conversionPattern value="%date [%thread] %-5level- %message%newline" />
            </layout>
       </appender>

     <appender name="DebugConsoleOutput" type="log4net.Appender.OutputDebugStringAppender">
         <layout type="log4net.Layout.PatternLayout">
             <conversionPattern value="%date [%thread] %-5level- %message%newline" />
         </layout>
     </appender>

  <root>
     <level value="DEBUG" />
     <appender-ref ref="ColoredConsoleAppender" />
     <appender-ref ref="DebugConsoleOutput" />
  </root>
  </log4net>
```

On to the project…

The brief for this article was pretty simple: "This is utterly self-serving, of course, but how about an addin to search ClarionMag?" Fortunately for Dave Harms and for me the implementation is also pretty straight forward. Rather than going into minutia of how everything in the addin system works it will probably be a nicer introduction to just get started and work through the project step by step.

The concept of solutions and projects should be familiar now since their introduction to Clarion 7 so fire up VS and create a new .Net Framework 2.0 project of the type Class Library and call it something like ClarionMagSearch. The first thing to do is setup the project properties. Right-click on the ClarionMagSearch project in the solution explorer and go to properties. Make sure the Target framework is set to .NET Framework 2.0 and Output type is Class Library.

Jump to the Build tab and set the output path. There are three pre-defined locations for addins that the Clarion IDE will automatically scan:

- `<Clarion install directory>\Bin\Addins\`
- `<Clarion install directory>\accessory\addins`
- `%appdata%\SoftVelocity\Clarion\7.0\`

There is also a method to define custom locations. You specify these in `%appdata%\SoftVelocity\Clarion\7.0\AddIns.xml`.

The most appropriate location for the addin I'm describing in this article is <Clarion install directory>\accessory\addins. I would leave the third option up to the Addin Manager if you deploy an "sdaddin" file, which is beyond the scope of this article and not necessary for the project I'm describing. (If you install SharpDevelop look in the SharpDevelop\doc\technotes directory for AddInManager.rtf, which has some info about sdaddin packages.)

Set the build output path to `<Clarion install`

directory>\accessory\addins\ClarionMagSearch

In the Build Events tab put <Clarion install directory>\bin\clarion.exe in the Post-build command line text box to launch the Clarion IDE after a build.

Next step is to create the addin file that tells the IDE what you want your addin to do. Each addin is specified using xml in a file with the extension .addin. The most important tags are AddIn, Manifest, Runtime and Path. Mostly they will be self-explanatory, for a fuller explanation see the tech notes distributed with the #develop source or take a look at existing addins definitions.

Right-Click on the project and choose Add | New Item of the type Text File. Name it ClarionMagSearch.addin. Click Add then open the new file for editing and paste in this code:

```
<AddIn name = "Clarion Magazine Search"
        author = "Brahn Partridge"
        url = "http://www.clarionedge.com/"
        copyright = ""
        description = "Perform a search on clarion magazine!">

  <Manifest>
    <Identity name="ClarionMagSearch" version="0.1"/>
    <Dependency addin = "SharpDevelop" version = "2.1"/>
  </Manifest>

  <Runtime>
    <Import assembly = "ClarionMagSearch.dll"/>
  </Runtime>

  <Path name = "/SharpDevelop/Workbench/Pads">
    <Pad id       = "ClarionMagSearch"
             category = "Tools"
             title    = "Clarion Magazine Search"
             class    = "ClarionMagSearch.SearchPad"/>
  </Path>

</AddIn>
```

Go to the properties for the file and make sure that Copy to Output Directory is set to Copy always or Copy if newer.

A Brief introduction to the tags:

<Addin> - The header of the addin definition, properties here define human readable information that will be displayed in the AddinManager. If you want, it is possible to query the addin tree from your addin and retrieve this information. The AddinManager is just another addin too!

<Manifest> - contains the identity and dependency information for your addin and is also used by the AddinManager. You can hard code your addin version here or use `@filename` to get the version information directly from the assembly.

<Runtime> - tells the IDE which DLLs to import when the addin is used, similar in concept to the win32 LoadLibrary. Note that the assemblies are only loaded when they are actually needed (first used) not at IDE startup. This design improves initial startup time of the IDE. If your addin requires other assemblies loaded, this is the place to import them.

<Path> - Represents the point in the addin tree that you wish interact with or extend. The addin tree is like a file system of objects within the IDE. The #develop AddinTree.rtf document describes it as a tree that "binds them all". It can be an existing path or, depending on your addin's purpose, it could be a new path you intend to create. Inside this tag you can do things like build menu items, create Pads or declare option dialogs. There are many different possibilities; this article will simply create a new workbench Pad. Have a look to see what else is being registered in this path by search the `<Clarion install directory>\Bin\Addin\*.addin files` for `/SharpDevelop/Workbench/Pads`. A lot can be learned about what goes where by examining the existing addin files!

Ok, so now the details of the addin have been specified everything is set to load a new pad into the IDE. Notice the `category = "Tools"` property in the `<Pad>` tag? There are a few smart helpers already built into the IDE for Pads. This category property will automatically add a menu item into the View | Tools menu. (See the #develop source code file `\SharpDevelop\src\Main\Base\Project\Src\Internal\Doozers\PadDoozer.cs` for more.)

This is probably a good time to actually do something and see what happens. Of course it is not going to work yet but it should be interesting to see what is going on. First start DebugView then build the solution in VS (F6). When the build is finished, if all goes well the Clarion IDE should be started and the ClarionMagSearch.addin definition loaded into the addin tree. Take a look at DebugView; there will be a bunch of output showing the startup process of the IDE but no mention yet of the ClarionMagSearch addin because of the lazy load feature. Back in the Clarion IDE go to the View | Tools menu and click on Clarion Magazine Search. Flip back to DebugView and there will be a few new entries (Figure 1).

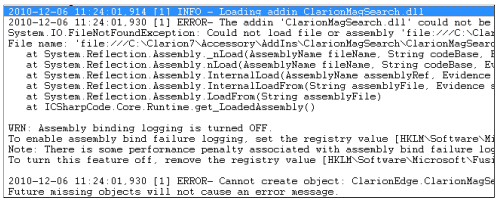

Figure 1. DLL loading error

The IDE attempts to load the DLL specified in the Runtime/Import section and since it doesn't exist yet it triggers an exception. Awesome!

Ok, back to VS, time to get some code written.

The standard layout for a #develop addin project is to put the .addin file in the root of the project and all source code under a sub directory called Src. Go ahead and set this up then create a new class, call it SearchPad (Figure 2).

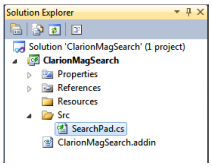

Figure 2. Creating SearchPad.cs

Paste the following code into SearchPad.cs:

```
using System.Windows.Forms;
using ICSharpCode.SharpDevelop.Gui;

namespace ClarionMagSearch
{
    class SearchPad : AbstractPadContent
    {
        SearchPadControl searchPad;
        static SearchPad instance;

        public SearchPad()
        {
            instance = this;
            searchPad = new SearchPadControl();
        }

        static public SearchPad Instance
        {
            get
            {
                return instance;
            }
        }

        public override Control Control
        {
            get { return searchPad; }
        }
    }
}
```

Notice how in the VS code editor that AbstractPadContent is underlined in red? The project needs to be able to reference some of the IDEs core ICSharp DLLs so that it knows what is going on. To fix this, in solution explorer right-click on references and choose add reference. Using the Browse tab find your `<Clarion install directory>\bin` folder

and select `ICSharpCode.Core.dll` and `ICSharpCode.SharpDevelop.dll`. Expand the References folder and for both the new items right-click and go to properties, set copy local to `FALSE` (Figure 3). If this setting is left at `TRUE` these assemblies will be copied into the addin folder during the build process which is not necessary.

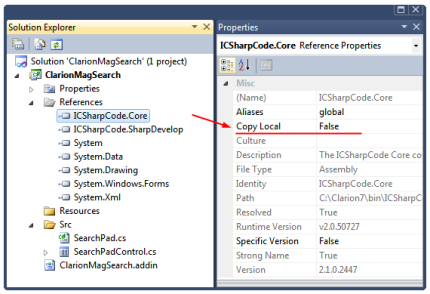

Figure 3. Setting Copy Local to False

The SearchPad class is the entry point for the AddinManager. The IDE requires that all Pads implement `AbstractPadContent` so that it (the IDE) can handle the basic housekeeping. What happens within the Pad is pretty much up to the designer. See how there is a `SearchPadControl searchPad` statement above the constructor? This is a UserControl that will contain everything needed to make the magic happen.

Creating the UserControl

In the Solution Explorer Src folder, Right-Click Add | User Control. Name it `SearchPadControl` and add three text boxes with labels for each, a checkbox and a button. I have named them `tbText`, `tbTitle`, `tbAuthor` for the text boxes (no such thing as an `ENTRY` control here!), `cbUseEmbeddedBrowser` and `buttonSearch` for the others.

Figure 4 shows what mine looks like.

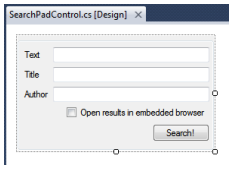

Figure 4. The SearchPadControl

For the impatient, hit F6 now to see how it looks in the Clarion IDE! (Make sure the IDE is not already open or the new DLL cannot be copied over).

If nothing happens don't panic! Go back to DebugView and see if there are any hints about what might have gone wrong. The IDE is set to handle most exceptions silently but you will see them in the debug output thanks to log4net.

The idea for this addin is pretty simple. The user enters a search term then clicks search. The addin will compile a URL from the text boxes that are not blank and start a web browser using the constructed URL.

First the button: In the design view of SearchPadControl go to the properties of the button (F4, just like Clarion!), click on the Events button and create a click event (Figure 5).

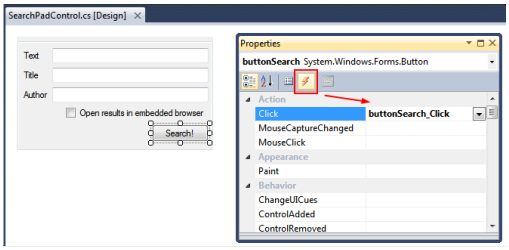

Figure 5. Creating a click event

This will open the code view for the new `buttonSearch_Click` method, paste in this code:

```
StringBuilder sb = new StringBuilder(String.Empty);
if (!string.IsNullOrEmpty(tbAuthor.Text))
{
    sb.Append("author:" + tbAuthor.Text);
```

```
}
if (!string.IsNullOrEmpty(tbTitle.Text))
{
    if (sb.Length > 0)
        sb.Append("+");
    sb.Append("title:" + tbTitle.Text);
}
if (!string.IsNullOrEmpty(tbText.Text))
{
    if (sb.Length > 0)
        sb.Append("+");
    sb.Append(tbText.Text);
}
if (sb.Length > 0)
{
    sb.Insert(0, "http://www.clarionmag.com/search/find?searchTerm=");
    if (cbUseEmbeddedBrowser.Checked == true)
    {
        FileService.OpenFile(sb.ToString());
    }
    else
    {
        System.Diagnostics.Process.Start(sb.ToString());
    }
}
```

The code is pretty simple. First a URL is constructed using a `StringBuilder`, making sure that only those values that have been filled in are appended, and then the URL is sent to a browser, using either the `System.Diagnostics.Process.Start` method which is just like using `ShellExecute` in Win32 or optionally open a browser embedded in the IDE using the `FileService.OpenFile` method.

Before I explain the FileService code, hit F6 and give it a go!

...

..

.

Awesome right?

There are a few helper services built into the #develop core code. While there isn't much documentation available for these as far as I could find, I learnt about them mostly from reading the #develop source and using Reflector. I also highly recommend is downloading and examining the #develop source code; although some of the core code has been altered by SV there is still a lot that is exactly the same. Getting to know Reflector can open up a world of examples!

Additionally, once you have the references to the `ICSharpXXX` assemblies in your project you can learn a lot via intellisense. For example, in the code view type `FileService.` (dot) and take a look at the methods available (Figure 6).

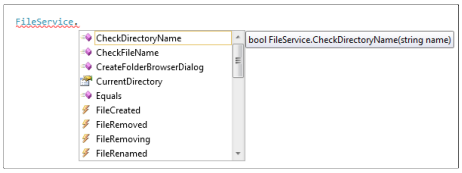

Figure 6. FileService methods

Other useful services available:

- `FileUtility`
- `LoggingService`
- `MenuService`
- `MessageService`
- `PropertyService`
- `ResourceService`
- `StringParser`
- `ToolBarService`

See `SharpDevelop\src\Main\Core\Project\Src\Services\` for the source.

It is really easy to persist settings for an addin by using the `PropertyService`. The default

for the `PropertyService` is to create an entry in `%appdata%\SoftVelocity\Clarion\7.0\ClarionProperties.xml` but there are also ways to define your own store location once you get more comfortable with addins.

After the call to `InitializeComponent` in the constructor load the saved value using this:

```
cbUseEmbeddedBrowser.Checked = PropertyService.Get("ClarionMagSearch.Options.useEmbeddedBrowser",
```

(the second parameter is the default value to use if the option is not already in the property store.)

To save the selection put this in a `CheckedChanged` event of the checkbox:

```
PropertyService.Set("ClarionMagSearch.Options.useEmbeddedBrowser", cbUseEmbeddedBrowser.Checked);
```

If all has gone well, you will now have your very own fully functional IDE pad to search for articles on ClarionMag!

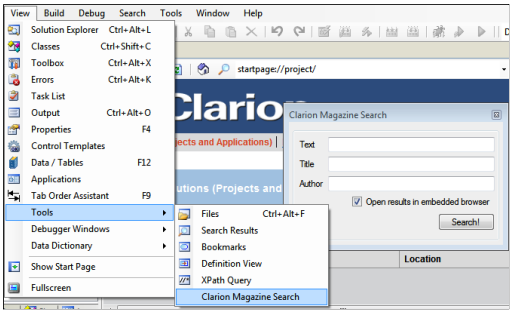

Figure 7. Searching ClarionMag from inside the C7 IDE

There's just one final step you may want to take: although Visual Studio has created the addin in place, to give someone else the addin you should create an sdaddin file. Just make a zip containing the .addin file and the DLL and change the extension of the zip file to .sdaddin. You can then install that addin via the IDE's AddIn Manager. To see how this works, download the sdaddin file from the link below and open the file in WinZip or any other ZIP tool.

My hope for this article is to give everyone an insight into the vast extensibility available with the new Clarion IDE. The more I dig around the more I discover is possible. SV have done a great job so far; with the addition of community and 3rdParty components (addins!), along with the AppGen and templates that give Clarion Win32 development such a huge bonus, maybe we can all retire early!

Further Reading:

- Visit my ClarionEdge site for other addins and articles. (Subscribe to the RSS feed to get notified of new addins as they are released!
- The folder doc/technotes in the SharpDevelop source code download contains an AddIn writing guideline and other information.
- SharpDevelop AddIn Writing Help
- Add-in Developer Quick Start Guide
- SharpDevelop 3rd Party Addins
- SharpDevelop screen recordings for developers
- ProgramArchitecture.pdf
- Free eBook: Dissecting a C# Application, Inside SharpDevelop
- Download the #develop source
- You can also use the git repository to checkout a copy of the source

Download the source

Download just the Clarion Search sdaddin file

Article comments

by Shane Vincent on April 4 2011 (comment link)

Nice job!

Tip of the Week: Two Ways To Look At Your Source

By Dave Harms

Posted March 31 2011

Last week's tip was by John Hickey, and this week's tip is something John pointed out to me when we were debugging a problem with ClarionTest. We were looking at a procedure in the embeditor and we really needed to see what was happening in two different places.

John said, "Split the window."

"What?"

"Split the window. Up there, on the Window menu."

So I looked on the Window menu, and sure enough there was a menu item called Split. I clicked on it and I saw something like Figure 1.

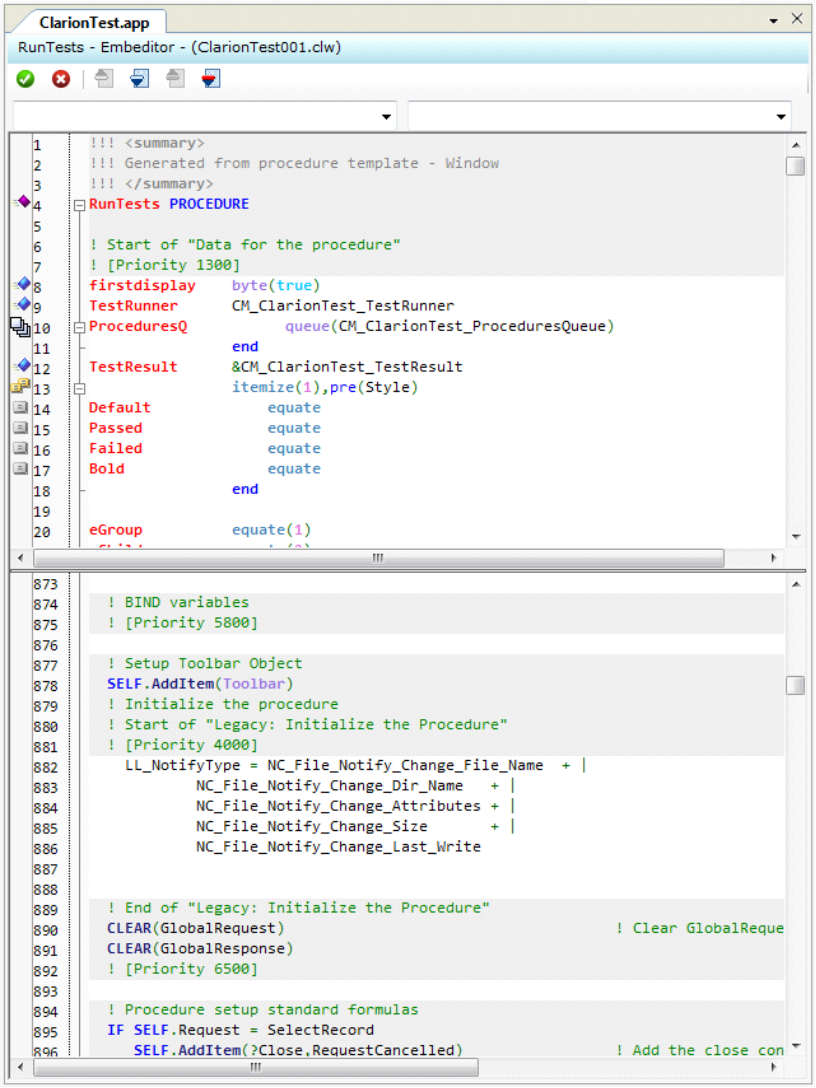

Figure 1. Splitting the embeditor window

In Figure 1 each pane is a separate view of the same text; you can navigate to different locations, or to the same location. If you're viewing the same part of the text file in both panes you can verify that changes made to one pane are instantly reflected in the other.

Having two separate views can be a big help when you find yourself jumping between two blocks of code to try to figure out what's going on.

I have no idea how long the Split command has been available (probably since the beginning) but I'd never ever even noticed it.

Split also works with regular text editing windows and even with the single embed editor.

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

Article comments

BACK TO TOP

Converting Between Clarion And Excel Dates/Times

By Pierre du Toit

Posted March 31 2011

Here are some useful facts about dates and times in both Clarion and Microsoft Excel.

Clarion time is expressed as hundredths of a second since midnight. The minimum time is 1, which is midnight. The maximum time is 8640000, which is 23:59:59.

Other Clarion time facts:

- 1 is equal to `8640001` `(24hr * 60min * 60sec * 100  + 1)`
- 1 second = `101` `(100 + 1)`
- 1 minute = `60*100+1=601`
- 1 hour = `60*60*100 + 1 = 360001`
- 1 day = `24*60*60*100 +1 =  8640001`

Excel time is expressed as a fraction of the total number of seconds in 24 hours. A time of 2:31:45 p.m. is equal to `((14 * 60 * 60) + (31*60) + 46)/86400`. The result is calculated to 19 decimal places.

Other Excel time facts:

- 1 second = `0.00001157407407407410`
- 1 minute = `0.00069444444444444400`
- 1 hour = `0.04166666666666670000`
- 1 day = `1.00000000000000000000`

A Clarion date is the number of days since December 28, 1800.

Other Clarion date facts:

- January 1, 1801 = Standard Date 4, the earliest valid Clarion date
- December 31, 9999 = Standard Date 2,994,626, the latest valid Clarion Date

An Excel date is the number of days since 1900/01/01, but for dates before March 1 1900 you have to subtract 1 because Excel thinks 29 Feb 1900 existed).

Converting a Clarion date to an Excel date:

- From 01 Mar 1900 to 31 Dec 9999: *`Clarion date`* `– 36161 = Excel date`
- From 01 Jan 1900 to 28 Feb 1900: *`Clarion date`* `– 36162 = Excel date`

Converting a Clarion time to an Excel time:

- `INT(Clarion time / 100) / 86400 = Excel time`

Combining dates and times

If you need to do time comparisons that span day boundaries, the best approach is the Excel way of storing the date as the Integer part and the time as the fractional part. (In Clarion circles this is sometimes called "start date".)

```
My_dateTime = today() + int(clock()/100) /  86400 ! no 100th seconds
```

```
My_dateTime = today() + clock() / 8640000         ! 100th seconds
```

Now date/time checks are simple. There is no need to check date then time then date. You will have no more "over midnight" mistakes.

You can also determine the length of an event by subtracting the earlier Date.Time value from the later one, provided you use the Excel format.

```
End_DateTime – Start_DateTime =  days.time
```

Notes on Clarion variables:

- `REAL` : 15 significant digits
- `DECIMAL` : The maximum length is 31

For a Clarion DateTime I recommend `DECIMAL(29,21)`
For an Excel DateTime I recommend `DECIMAL(27,19)`

Clarion date/time masking for reports/Browse fields

Often I want to express 1 day 1 hour as 25:00:00, but Clarion only understands time from 00:00:00 to 24:00:00. So I need to define a little tweak, using an Excel date/time:

```
justTIME = (ExcelDateTime - int(ExcelDateTime)
tweekHour = INT(justTIME * 24)
tweekMin = INT(justTIME * 24 *60 - (tweekHour *60))
tweekSec = IMT(justTIME * 24 *60 *60) - (tweekHour *60 *60) - tweekMin*60
! now add the days as hours
tweekHour += int(ExcelDateTime)*24
! formating to a string variable
newTime = format(tweekHour,@p<l<#:p) & format(tweekMin,@p##:p) & format(tweekMin,@p#
```

Now there is another way (as always, there are 101 ways to skin a cat):

```
! Get real Clarion Time format (add 1 in case ExceldateTime format i.e. no 100th sec
! Remember 101 in Clarion time = 1 sec --- 100 is less than 1 second
justTIME = (ExcelDateTime - int(ExcelDateTime) * 8640000 + 1 ! add 1 in case Excel
newTime = format(justTime,@t04) ! hh:mm:ss
I# = newTime[1:2] + int(ExcelDateTime)*24    ! slice hours and add days as 24 hours
newTime = format(I#,@p<<<#p) & newTime[3:8]  ! max hours = 999 = 41 days 15 hours
newTime = format(I#,@p<<<<#p) & newTime[3:8] ! max hours = 9999 = 416 days 15 hours
```

Clarion DATE/TIME storage

Clarion developers commonly store dates and times as LONGs. But Clarion also has `DATE` and `TIME` types. While these appear to be `LONG`s, they are actually stored as follows:

```
DATE    YYYYMMDD
TIME    HHMMSShh
```

You can convert a `DATE` to the individual fields using `OVER`ed groups:

```
myDate      DATE
myDateGroup GROUP,OVER(myDate)
myMM          BYTE
myDD          BYTE
myYYYY        SHORT
            END

myTime      TIME
myTimeGroup GROUP,OVER(myTime)
```

```
myHrs          BYTE
myMin          BYTE
mySec          BYTE
myHsec         BYTE
             END
```

I hope you find these date/time tips as useful as I have.

Editor's note: Pierre's article was used as the subject of a unit testing exercise at a recent ClarionLive! webinar by Dave Harms and John Hickey. Read more here.

Article comments

BACK TO TOP

Unit Testing Webinar Workshop Takes On Dates/Times

By Dave Harms

Posted March 31 2011

On Friday, March 25 2011 John Hickey and I conducted a ClarionLive! webinar workshop on unit testing with ClarionTest. So far just a handful of Clarion devs have jumped on the unit testing bandwagon, and we wanted to get a bunch more to join us.

In this article I'll provide an overview of what we did during that webinar and some of the things we learned about the unit testing process under Clarion. I'll also talk about some of the improvements we're making in ClarionTest as a result of that webinar.

And now that ClarionTest is rounding into shape, I also hope to have an article up in April on how it actually does what it does.

A word about unit testing and ClarionTest

Unit testing isn't exactly a new idea, but development platforms (like .NET) make it particularly easy to automate the unit testing process. I've had some very rewarding experiences with unit testing and C# (using nUnit), and I wanted to bring some of that functionality to the Clarion world.

If you haven't read about ClarionTest yet, you might want to start with Start Testing Your Clarion Win32 Code With ClarionTest, an article I published in December 2010. That article also provides a brief introduction to some unit testing concepts.

In February 2011 I did a ClarionLive webinar on unit testing, and following that webinar John started using ClarionTest for some of his SQL-related work. John's made some really nice enhancements to ClarionTest, which he and I will explain in that upcoming article. If you read my December article and/or view the February webinar, you'll be well-prepared for the remainder of this article.

Getting the workshop participants ready

The March 25 webinar wasn't really a standard webinar where a presenter showed how to

accomplish some task; instead we wanted participants, people who would walk through the unit testing process with us. About a half dozen developers got on board, and we began with the installation process.

So step one was to get over to the Google code repository and download ClarionTest. There are usually two downloadable zips for each release; the only difference between them is that one contains all of the DLLs and EXEs; if you download the other zip you need to build ClarionTest before you can use it.

Chances are you're going to want the bundle with all of the DLLs and EXEs.

And most people don't need all of the source code that goes with it. Heck, probably all you really need is ClarionTest.EXE, some libsrc files and a template. This was one of the best things about going through the workshop; realizing that I'd put a whole bunch of stuff up there that no one really needed. John and I also agree that what ClarionTest really does need is a good installer. We're working on it (and it will be powered by SetupBuilder, of course).

Once everyone had downloaded the zip we set about making sure the installations were all correct. This involves the following steps (most of which could, ahem, be handled by an installer):

- Copy ClarionTest.exe and its DLLs to an install directory
- Copy the libsrc files to a suitable libsrc directory
- Copy the template to a suitable template directory
- Register the template
- Run ClarionTest against the supplied test demo DLL.

Actually that last test could easily be handled by SetupBuilder as well.

Paths with spaces

Everyone got the ClarionTest files unpacked and installed without too much difficulty. But several participants had difficulty running ClarionTest against the demo DLL.

We uncovered two problems that could cause ClarionTest to fail, and both were related to how we were executing ClarionTest.

There are currently two main ways to run ClarionTest. One is to launch it once, tell it to monitor a directory, and then either let ClarionTest execute tests whenever a test DLL has changed or click a button to execute tests on the automatically-reloaded DLL. The other way, and the one we chose for the webinar, is to run ClarionTest as a post-build task.

I still use this latter approach most of the time because I'm accustomed to the workflow. And for a webinar presentation on a single monitor it made more sense, since after a build ClarionTest pops up and runs the tests in the just-built DLL. That meant I wouldn't constantly have to Alt-Tab during the webinar to see the test results. If I were developing on

two monitors, then launching ClarionTest individually on one of the monitors and doing my development on the other would probably make more sense.

To run ClarionTest as a post-build task you use a command something like the following:

```
..\ClarionTest.exe demodll.dll /run
```

The downside of this approach is that it if ClarionTest encounters a problem, the error it reports may not be all that helpful.

Several participants had UAC enabled, and the first time you run an unsigned EXE under UAC you'll be asked to give the application permission to run, either just this time or for all future requests. Since you'll be running ClarionTest many times you'll want the latter option.

The second problem we had was that several participants had placed ClarionTest in a folder that resulted in a path with a space, such as

```
..\..\ClarionTest 2.0\ClarionTest.exe demodll.dll /run
```

That also caused an error on the post-build task. The answer was to include the path in quotes:

```
"..\..\ClarionTest 2.0\ClarionTest.exe" demodll.dll /run
```

A third possible problem, which did not show up in the webinar, is that ClarionTest may not be able to find a DLL needed by the test DLL. For this reason we may need to restore ClarionTest's very first way of loading DLLs, which was via an entry field and a file dialog for choosing the DLL name. This approach generally gives the most useful information in the event that ClarionTest cannot load the test DLL or run the tests.

Creating a test DLL

At this point we had everyone able to run ClarionTest against the demo DLL. Woohoo!

The next major hurdle was creating a new test DLL. A test DLL has just two simple requirements:

1. It must be a DLL (and there's a trick here)
2. It must have the ClarionTest global extension loaded

Creating a DLL in Clarion 7 takes one additional step more than in C6, because of a C7 bug. First, you create a DLL as in Figure 1.

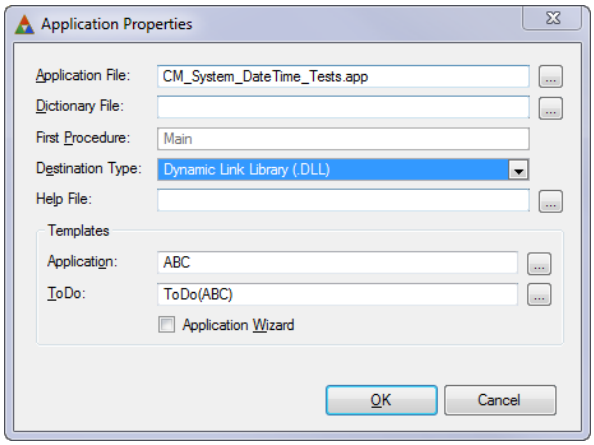

Figure 1. Creating a DLL.

But as of C7.3.7995 there's a bug in the project data. Even though you created the app as a DLL, the project data will show the output type as EXE. You have to set the output type to DLL (Figure 2).

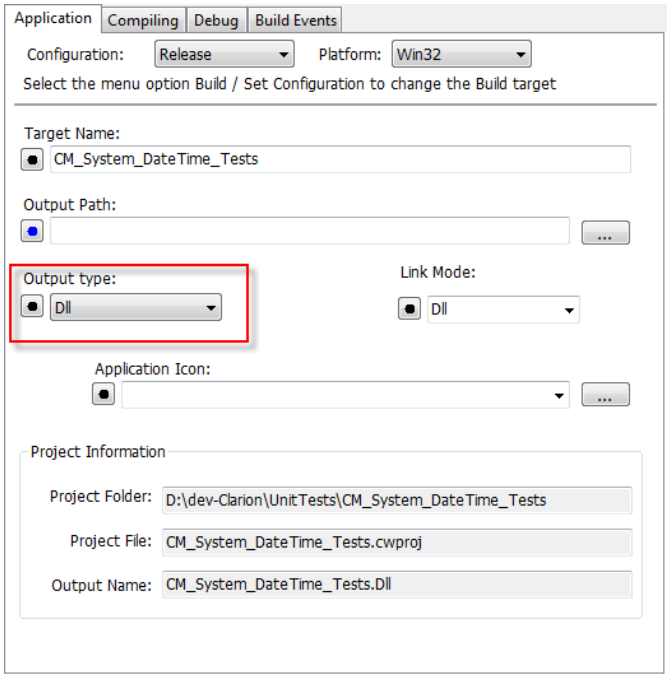

Figure 2. Setting the output type to EXE

And finally add the global extension, as shown in Figure 3.

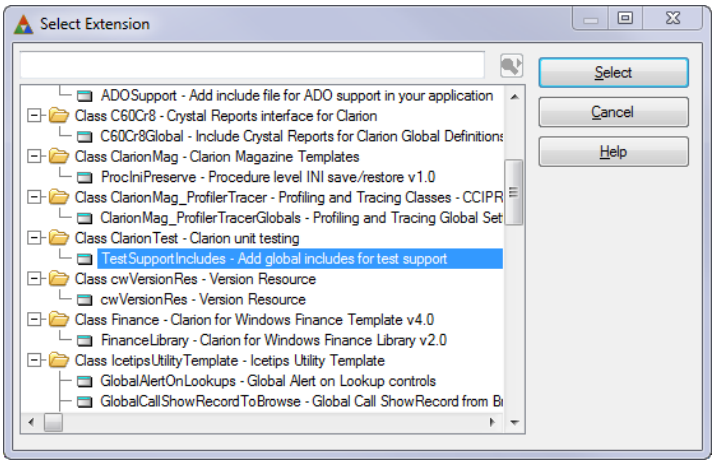

Figure 3. Adding the global extension.

Creating a test procedure

When creating a test procedure, always choose the procedure template from the Defaults tab (Figure 4).

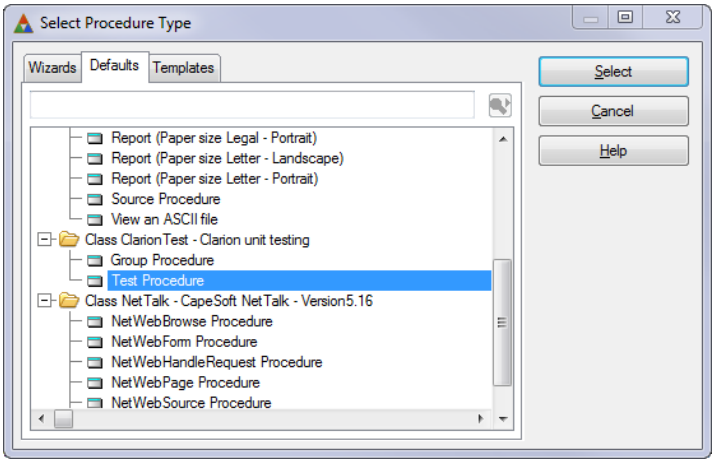

Figure 4. Creating a test procedure

I won't go into the details of how test procedures work - please read Start Testing Your Clarion Win32 Code With ClarionTest for more information.

In the webinar we created a test procedure and verified that it compiled. We also set up a post-build task to run ClarionTest after every build. Since empty tests pass, we made sure that everyone could get a successful test result after building the DLL.

Then we started *breaking* things, using a test-first methodology.

What to test?

Our participants finally all had their test environments ready for some actual code. I'd recently received a submission from Pierre du Toit describing how to convert Clarion dates

and times to Excel dates and times, so I suggested that as a starting point. I knew we wouldn't have time to get through all of Pierre's points, but I figured we should at least be able to make a good start. You may want to read Pierre's article at this time to get a feel for where the code was headed.

Test-driven development

Test-driven development, or TDD, is one of many different ways to do development. As its name suggests, TDD is all about writing the test *before* you write the code. I particularly like TDD for writing classes because it forces me to start by thinking about how I want to use the class rather than how I want to write the code.

When you write the test even before you write the class, your test is inevitably going to make use of at least one class and one or more methods that don't even exist. So naturally it's not going to compile.

I didn't keep a copy of the exact code we started out writing, but it looked something like this:

```
DateTime        class(CM_System_DateTime)
    code
      DateTime.SetDate(Date(3,25,2011)) ! TestClarionDate
      AssertThat(DateTime.GetExcelDate(),IsEqualTo(|
        DateTime.GetDate() - 36161),'Wrong Excel date')
```

The next thing we needed was a bare-bones class. There are a couple of ways to approach this, but a useful bare minimum, with stubs for the required methods, is as follows. First, the CM_System_DateTime.inc file:

```
CM_System_DateTime  class,type,module('CM_System_DateTime.clw'|
                      ),link('CM_System_DateTime.clw')
CurrentDate             long,protected
SetDate                 procedure(long newDate)
GetExcelDate            procedure,long
                        end
```

And here is the CM_System_DateTime.clw file:

```
                        member()

                        MAP
                        end

      include('CM_System_DateTime.inc'),once
```

```
CM_System_DateTime.SetDate  procedure(long NewDate)
     code
     self.CurrentDate = newDate


CM_System_DateTime.GetExcelDate     procedure!,long
     code
     return -1
```

At this point the class compiled, but the test failed because I'd written `GetExcelDate` to return what I was pretty sure was an invalid value (the method has to return something). And with that we were finally off to the races.

We spent the next hour or more writing test code, refactoring the class, and automatically running all the tests on every successful build. In the end we only got a few more methods written (see Figure 5), but we had a lot of interesting discussions about class design and best practices along the way. For more on what we accomplished please download the source and have a look through the test procedures. That's another benefit of unit testing - not only does it verify that your code *works*, but the unit tests serve as documentation for how your code should be *used*.

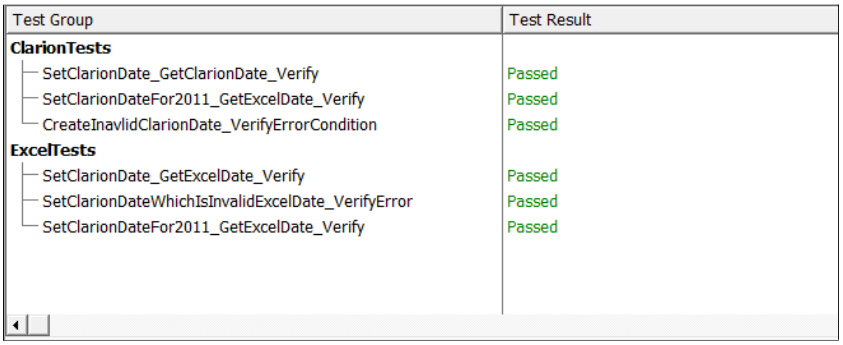

Figure 5. Successful tests created during the webinar

Summary of steps

During the webinar we took the following steps to achieve successful unit testing:

1. Downloaded and installed ClarionTest
2. Copied the necessary libsrc files to an appropriate location
3. Copied the template file to an appropriate location
4. Registered the template
5. Verified ClarionTest's operation using the test demo DLL
6. Created a new unit test DLL
7. Add the global extension to the DLL

8. Changed the output type of the test DLL from EXE to DLL
9. Created an empty test procedure
10. Verified that ClarionTest worked with the test DLL
11. Wrote some test code (breaking the compile)
12. Created (later, modified) a class to fix the compile errors
13. Ran the test, which failed
14. Wrote code to make the test pass
15. Wrote more unit tests, refactored some methods, ran unit tests on each build

Conclusions

It took a lot longer than I expected to get our workshop group to the point where we were finally ready to write some code. For one thing, the current install process for ClarionTest is too cumbersome. For another, there were a couple of problems (UAC, paths with spaces) that neither John nor I had previously encountered, and it took us a little while to figure out what was happening. But the biggest issue is simply the number of steps currently necessary to get ClarionTest up and running.

SetupBuilder should be able to handle steps 1 through 5 with an installer, so consider that a single step in the future.

Steps 6 through 8 probably won't change a whole lot. Step 7 is a bug that will hopefully be fixed by SV one day. Steps 10 through 15 are the essential discipline of test-driven development, and for the most part can't be automated. But John has come up with a nifty way to create the initial class INC and CLW files, which will make step 11 a whole lot easier. More on that in a future article!

Optimally I think we can achieve a one step install (with initial test), a four step process for creating and testing a unit test DLL, and a four step cycle for creating and eventually passing a unit test. Throw in some ongoing refactoring (made much easier because you're continuously testing your changes) and you have a unit testing protocol for Clarion!

Feel free to add more unit tests to the app, and please send them back to me so I can integrate them into the downloadable source.

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).

Article comments

BACK TO TOP