Clarion Magazine

Clarion News

- » SV To Explain Architecture, Demo Clarion.NET CodeGen Process In Oz
- » Software Developers Profile Exchange Updated
- » Oz DevCon Schedule
- » Noyantis Acquires gSec
- » BSRef 1.2 Application Analyzer Suite
- » iQ-Sync 1.08
- » BoTran2 1.1
- » EZChangeLog Reporter 1.7
- » SetupBuilder 7.0 Build 2712

[More news]

- » Clarion.NET FAQ
- » Clarion# Language Comparison

[More Clarion & .NET]

[More Clarion 101]

Latest Free Content

- » Source Code Library 2009.08.31 Available
- » Friday ClarionLive Webinar: SQL Super Panel

[More free articles]

Clarion Sites

Clarion Blogs

Latest Subscriber Content

Source Code Library 2009.08.31 Available

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

Posted Thursday, September 10, 2009

Superfiles and NAME

Having covered Superfiles in the previous episode, Steve Parker tackles the intricacies of how to set arbitrary names for the tables inside Superfiles.

Posted Monday, September 14, 2009

Clarion 8 Wish List: Overview and Comments

This summer Clarion Magazine asked for your wish list of C8 features. There were a number of responders including several who objected to the whole idea of a C8 wish list on the grounds that Clarion.NET is the future. Dave Harms reviews and comments on the wish list, and offers his opinions on the Win32 vs .NET controversy.

Posted Thursday, September 17, 2009

Writing Your Own Template Chain: Creating Embeds At Runtime

The Clarion template language has some impressive capabilities, and one of the most useful is the ability to create embed points on the fly.

Posted Monday, September 21, 2009

Friday ClarionLive Webinar: SQL Super Panel

Clarion Magazine's Dave Harms will be moderating the first ever ClarionLive SQL Super Panel, featuring David Swindon, Jim Morgan, Joe Tailleur, and Shawn Mason. Topics include: Definition of SQL; The range of SQL approaches; The SQL perspective; The Clarion perspective; Advantages/Disadvantages as compared to TPS; Questions you need to ask yourself; and more. And stay tuned for a special announcement at the end of the webinar!

Posted Thursday, September 24, 2009

Template Logging/Debugging Revisited

There really is no such thing as a template debugger, much as that would make life easier for many Clarion developers. But thanks to Mark Goldberg and Russ Eggen, there is a nifty technique for logging template output that makes template debugging much easier. Dave Harms revisits the code in C7 and explores some of the advantages of using DebugView to log messages, whether those messages come from templates or from regular source code.

Posted Friday, September 25, 2009

The Problem With Embeds, Part 1: What Went Wrong?

As Clarion developers we (well, most of us) live and die by embed code. Embeds are, after all, at the heart of Clarion's effectiveness. They're the primary way we add custom code to applications that are otherwise template-generated. And if you're like most Clarion developers, you're probably using embed points the wrong way. Without realizing it you're making your applications more difficult to maintain, harder to debug, and almost impossible to document.

Posted Monday, September 28, 2009

Making TPS Superfiles

In previous articles Steve Parker explained what TPS superfiles are and how to name them. In this installment Steve shows how to create superfiles from existing TPS files.

Posted Monday, September 28, 2009

AES Encryption And Test Vectors

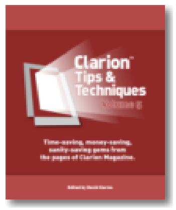
AES encryption is easy to implement in Clarion, thanks to Carl Barnes' example from a few years back. But how do you ensure the encryption is working correctly? With test vectors, of course.

Posted Wednesday, September 30, 2009

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

Source Code

Printed Books & E-Books



From The Publisher

About Clarion Magazine

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

Subscriptions

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

Satisfaction Guaranteed

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

Dave Harms

ISSN

Clarion Magazine's ISSN

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

About ISSN

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

Clarion Magazine

Clarion News

Search the news archive

Clarion Third Party Profile Exchange October 1 2009 Release

The Clarion Third Party Profile Exchange has been updated. This is a web updated release - data version.

Posted Monday, October 05, 2009

SkinFramework Wrapper Template 1.04

Version 1.04 of the SkinFramework Wrapper template is now available. The new version has been uploaded to members area along with a new demo example. Changes include: Exclude Modules options now specified prior to the LoadSkin method call; New sample skins added to demo application. The new version can be downloaded from the Members area using the original download and registration details contained in your sales email.

Posted Monday, October 05, 2009

SetupBuilder 7.0 Build 2734

Lindersoft has released SetupBuilder 7.0 Build 2734. This release is available, free of charge, to all SetupBuilder customers who have an active SetupBuilder maintenance subscription plan. The update contains some important bug fixes and improvements.

Posted Monday, October 05, 2009

CFC Library 3.0

CFC Library 3.0 is now available. The library consists of a set of classes and templates for creating tooltips, processing the hot keys and for creating the WinAPI menus and toolbars.

Posted Monday, October 05, 2009

ClarionLive Aussie DevCon Daily Reports

ClarionLive will feature daily reports from the Aussie DevCon daily at 8 p.m. Pacific time during the conference, starting Sunday Oct 11.

Posted Monday, October 05, 2009

EasyListView 1.00

EasyListView is a Clarion wrapper around a .NET ListView. It makes the ListView easy to use in your Clarion applications and provides some extra functionality. Features include: ; Automatically transforms data source (supports XML files, SQL connections, Clarion queue and Clarion Database Drivers) into a fully functional ListView, including automatically sorting and grouping rows; Supports owner drawing, including rendering animated graphics and images stored in a database; Supports 5 ListView views (SmallIcon, LargeIcon, List, Tile, Detail); Supports automatic grouping; Supports collapsible groups (Vista only); Columns can be fixed-width, have a minimum and/or maximum width, or be space-filling (Column Widths); Displays a "list is empty" message when the list is empty (obviously); Supports alternate rows background colors; Supports searching (by typing) on any column; Supports custom formatting of rows; Easily edit the cell values. All these are template driven. No third party library or components needed. Requirements: * Clarion 6.3 / Clarion 7; ABC or Legacy Template Chains; 32-bits only; .NET Framework 2.0 and above; Windows XP and above. Demo available. Price: $95 (New Subscription: 1 Developer license + 1-year Maintenance Plan). Release date: Oct 01, 2009 (will be shipped in October 2009).

Posted Monday, October 05, 2009

iQ-XML 2.53

iQ-XML is now available. Changes include: Fixes to the meter bar; Ability to double-click on any of the result queue's for example copy to/from information; Megs show as "MB" instead of a large bytes number.

Posted Monday, October 05, 2009

SV To Explain Architecture, Demo Clarion.NET CodeGen Process In Oz

At this year's Oz DevCon in Eden, Australia, SoftVelocity's Bob Zaunere will for the first time explain the architecture of the new templates being developed for Clarion.NET. Z will also demonstrate the Clarion.NET application generation process.

Posted Wednesday, September 23, 2009

Software Developers Profile Exchange Updated

The Software Developers Profile Exchange has been updated. This is a abbreviated online and web updated release - data version. Anyone can freely access the Abbreviated Online Version.

Posted Friday, September 18, 2009

Oz DevCon Schedule

The Oz DevCon schedule is almost final. The 3rd Party Suppliers are continuing to contribute and have donated towards $16,000 worth of product with more to come on board. A number of people are coming from overseas including USA, South Africa, UK, Lithuania, New Zealand, and Vanuatu.

Posted Friday, September 18, 2009

Noyantis Acquires gSec

Noyantis Software announces the acquisition of the gSec 3rd party security template. The gSec product was originally created by Gitano Software, then purchased and updated by Wingnut Solutions before coming to Noyantis. During the coming weeks, the template will be reviewed and brought inline with the other Noyantis templates. At that point it will be available for purchase via the Noyantis web site, Clarionshop and Motleysoft. Some upgrade options for existing customers will be available. Enhancements and new options are planned for the future.

Posted Friday, September 18, 2009

BSRef 1.2 Application Analyzer Suite

Comsoft7 is releasing BSRef Application Analyzer Suite version 1.2. BSRef is a combination template and application to analyze your application.

Posted Friday, September 18, 2009

iQ-Sync 1.08

iQ-Sync 1.08 - Enhanced iQ-Sync now to handle both GETTING and PUTTING file(s) on an FTP server. If you are copying or backing up the same files daily, weekly, monthly - or going home from work each night forgetting to copy your work off to your server, or it's just a pain in the butt -- use iQ-Sync. It's totally free. Simply setup an unlimited number of Copy projects, and let iQ-SYNC handle it for you. Copy files to your server, production or testing servers, or your backup/ USB devices quickly and easily.

Posted Friday, September 18, 2009

BoTran2 1.1

BoTran2 1.1 adds a new export wizard to find and export text from the embeds to the language file.

Posted Friday, September 18, 2009

EZChangeLog Reporter 1.7

EZChangeLog Reporter v1.7 provides additional report and data query capabilities, including output to spreadsheet for EZChangeLog Professional Edition data files.

Posted Friday, September 18, 2009

SetupBuilder 7.0 Build 2712

Lindersoft has announced SetupBuilder 7.0 Build 2712. This release is available, free of charge, to all SetupBuilder customers who have an active SetupBuilder maintenance subscription plan. The update contains some important bug fixes and improvements.

Posted Friday, September 18, 2009

ClarionLive Webinar #26 - Mark Goldberg

ClarionLive Webinar #26 features Mark Goldberg, an expert Clarion hand coder with over 20 years of experience. This will be a more interactive webinar, as Mark covers a variety of topics including: Create Event Listeners with REGISTER() - but why?; A step towards demystifying classes in Clarion; Using OutputDebugString to debug without using the Debugger (or log files); And more. Come with your curiosity and questions.

Posted Thursday, September 17, 2009

ClarionLive Weekly Webinar #24 - Robert Paresi on iQ-Sync and Bruce Johnson on OddJob and WinEvent

The pre-Labor Day weekend ClarionLive Webinar features two new products, OddJob and iQ-Sync, plus the more mature WinEvent.

Posted Thursday, September 03, 2009

iQ-Sync

Robert Paresi's iQ-Sync is a little utility which simply copies and synchronize files. Simply setup an unlimited number of copy projects, and then when you need to execute it, highlight the project and press Run. iQ-Sync will notify you if files are in use, and can optionally wait until they are available. iQ-Sync is free.

Posted Thursday, September 03, 2009

LogFlash Half Price

The LogFlash audit trail and undelete templates are available at half price until September 8th. The cost during the sale is $97.50, with no annual maintenance charges. 60-day guarantee, demo available. To order use the promotion code "LogFlash 50% off".

Posted Thursday, September 03, 2009

Aussie DevCon Training Schedule

The schedule for the Aussie DevCon Training has now been determined, and includes Clarion.NET training by Pierre Tremblay from SoftVelocity. Due to the relative lateness of determining the training schedule the deadline for the Regular Early Bird Rate has been extended to September 13th.

Posted Thursday, September 03, 2009

Handy Zip'N Email and Handy Zip'N FTP

New versions of Handy Zip'N Email and Handy Zip'N FTP. Both apps are still unlimited. These new versions provide for up to nine configurations which can be recalled with a single click. The CHT query control used by both apps now includes autocomplete capability for easier use. Query keywords pop into place automatically after you type three or more matching characters. Both apps are pre-compiled versions of how-to-demos shipped in source .APP format with the CHT toolkit and can be built with either Clarion 6 or Clarion 7 and The Clarion Handy Tools.

Posted Thursday, September 03, 2009

Product Scope 7.9

Product Scope 7.9 is now available. Product Scope 7 provides a combination of tools for keeping track of comparative product information, places to buy, manufacturers, product images, located on the Internet and on local storage drives. Updated features include: Product Scope 7 Clipboard; Spreadsheet Named Output. 30 day trial available. Product Scope 7.9 is a major change from previous installs - previous Product Scope 7 users (7.5.337 or earlier) need to use the conversion program and follow the detailed instructions in the PDF, available at http://www.encouragersoftware.com/download.html. Product Scope 7.8.377 - use full install or web update method.

Posted Thursday, September 03, 2009

CommandBars Wrapper Template 2.00

Noyantis Software has released CommandBars Wrapper template version 2.00. This release is available in the members area in the web site; a new demo is also available. Modifications include: Codejock v13.1.0 compatibility added; 'Event Handler' facility added; 'Initial State' of controls can now be conditionally set; 'Initial State' of bars can now be set; 'State" of control can now continually mimic a standard clarion control; Alignment property added to control definition; Standard Windows Actions added (STD:Close, STD:Cut, STD:PrintSetup etc); Standard Clarion Actions now handled for bars that contain Clarion menu content type; Min / Max / Close buttons added for MDI Windows; More control types added to System Menu; Icons now automatically added to Project List (if filename prefixed with a "~"); Default Font attributes now amendable; Office 2007 Theme files can now be applied; Office 2007 Window Frame can now be applied; RibbonBar Tabs can now be hidden; Button styles can now be set per Bar in one command; 'Show Gripper' option added to

bar definition; 'Show Expand / Customize' option added; 'Default Tooltip Style' option added; Extra Embed Points added; ImageRsc storage variable increased; BUG FIX: Save & Restore of Window locations corrected; BUG FIX: Height of bar area incorrect after bar minimized or moved (Application Frame Extension); BUG FIX: Popups were still being displayed when the control had lost focus. The new version can be downloaded from the Members area using the original download and registration details contained in your sales email.
Posted Tuesday, August 25, 2009

EZChangeLog Reporter 1.7

EZChangeLog Professional 1.7 is now available. The main features of EZChangeLog Reporter add additional report and data query capabilities, including output to Microsoft Excel spreadsheets for EZChangeLog Professional data files. Version 1.7 adds custom reports. Use a simple, intuitive interface to quickly create and save an unlimited number of custom reports.
Posted Tuesday, August 25, 2009

CHT Build 13C1.00

CHT Build 13C1.00 has been released. Current CHT subscribers can start their Webupdaters (for C6 and C7) to bring their installations up to date with the latest code.
Posted Tuesday, August 25, 2009

EasyListView Version 1.00

An updated demo is available for EasyListView Version 1.00. Features include: New SQL connections and Clarion Database Drivers (tps) examples; Forecolor, Backcolor, Alternate row color (greenbar) selections; Images from BLOB / Image files; Example uses MDI; Added possibility for graphic data like ProgressBar and MultiImage; Added possibility to catch Double-Click and Right-Click on ListView. EasyListView is a Clarion wrapper around a .NET ListView. It makes the ListView easy to use in your Clarion applications and provides some neat extra functionality.
Posted Tuesday, August 25, 2009

Capesoft OddJob In Beta

From the folks that brought you easy access to Services comes easy access to Windows Job Objects. A Job Object allows groups of processes to be managed as a unit. OddJob allows you to start a process and "attach" both the Standard In, and Standard out from that process to your process. You can feed it the input it requires, either on the command line or via Standard In, and you can also capture the output via Standard Out. So there's no more black window, and there's no more having to guess the result. Because you can capture, and examine, Standard Out, you can be sure of exactly what happened. You also control the "Environment" that the other process has to run with. So you have complete control over the environment variables that the other process will "see" as it runs. OddJob also allows you to monitor the processes in a Job returning information like CPU and Memory usage. So if you attach a process to your job, you can then decide if the other process has consumed too many resources, and if necessary, kill it. And restart it. So if you want one program to monitor another, this makes an excellent way to do that. Better yet, if the process being monitored is something you wrote, then you could be more polite than just killing it, by sending it a message to shut-down or something. So another way to think of OddJob is as a process manager. OddJob was originally written to manage PHP scripts in conjunction with NetTalk 5. OddJob costs $79, but is currently on special for $59 while in beta.
Posted Tuesday, August 25, 2009

RADFusion International and Combit have severed ties

Combit and RFI are no longer doing business together. Russ Eggen will continue support on all code RFI has published.

Posted Tuesday, August 25, 2009

Clarion ProImage 2.1

Clarion ProImage 2.1 with live video image capture has been released. ProImage is a drop in Image Editor for Clarion programmers. It allows you to add full featured photo processing and image editing to your application in less than 10 minutes. It is available for Clarion 7.x, Clarion 6.x and Clarion 5.5G. Both ABC and Legacy templates are included. This is a free upgrade for anyone who purchased ProImage on January 1, 2008 or later. Developers who purchased before that date can upgrade for $59.95 USD. Note: Since version 2.0 of ProScan is also available, developers who own both products can save an additional $20 USD by upgrading both at the same time for $99.95 USD. The full version of ProImage is $249.95 USD and there is a discount for buying it as a bundle with ProScan.

Posted Tuesday, August 25, 2009

Clarion Magazine

Superfiles and NAME

by Steven Parker

Published 2009-09-14

In the last action packed episode, I talked about TopSpeed superfiles. I talked about what they are, how they differ from standard TPS files (virtually not at all) and why some developers shy away from using them.

In this installment, I will look at creating and using superfiles. I will also explore the degree to which I can use the Name attribute to put a superfile under full programmatic control.

Basics

I start with the TopSpeed Database Driver help topic, "Storing Multiple Tables in a single .TPS File." This topic tells me:

> By using the characters '\!' in the NAME() attribute of a TopSpeed file declaration, you can specify that a single .TPS file will hold more than one table. For example, to declare a single .TPS file 's&p.tps' that contains 3 tables called supp, part and ship:
>
> ```
> Supp FILE,DRIVER('TopSpeed'),PRE(Supp),CREATE,NAME('S&P\!Supp')
> ...
> Part FILE,DRIVER('TopSpeed'),PRE(Part),CREATE,NAME('S&P\!Part')
> ...
> Ship FILE,DRIVER('TopSpeed'),PRE(Ship),CREATE,NAME('S&P\!Ship')
> ...
> ```

I want to reiterate what I think is a very important point. When I make an entry in the Name prompt in the Dictionary Editor, I am actually creating a Clarion *Label*.

A Label is how I refer to an entity, a file/table, a procedure, a method, a routine or a variable in code. Please take a moment to read the online help on "Label."

When I make an entry in the Dictionary Editor's Full Pathname prompt, I am actually placing a value in the Name attribute. There are a number of articles in Clarion Magazine on the subject of the Name attribute. I encourage you to take a moment and review one or more until you are comfortable with Name and what you can do with it.

Normally, it is permissible to leave the Full Pathname (Name) blank. When Name is left blank, Clarion uses the Label (which may never be blank) for the Name. In this case, the only manipulation available at runtime is System{Prop:DataPath} to set the physical location of the file. (See Name Becomes Irrelevant for more information on Prop:DataPath.)

Superfiles are the exception. To create a superfile, not only is an entry in Full Datapath required but that entry must follow specific syntactic rules, as described in the documentation above. In fact, it is the use of this syntax that tells the Clarion runtimes that I am using a compound file.

Creating a basic superfile

The first step is to create the files in the dictionary. At this point, just fully specify the files that will eventually go into the superfile.

> **Note:** The demo app for this section of the article is SuperFiles.app and SuperFiles.DCT. You may download them at the end of the article. The zip contains both C5.5 and C7 versions.

With that done, decide on the DOS file name of the superfile. That is, decide on the name that will appear in Windows Explorer.

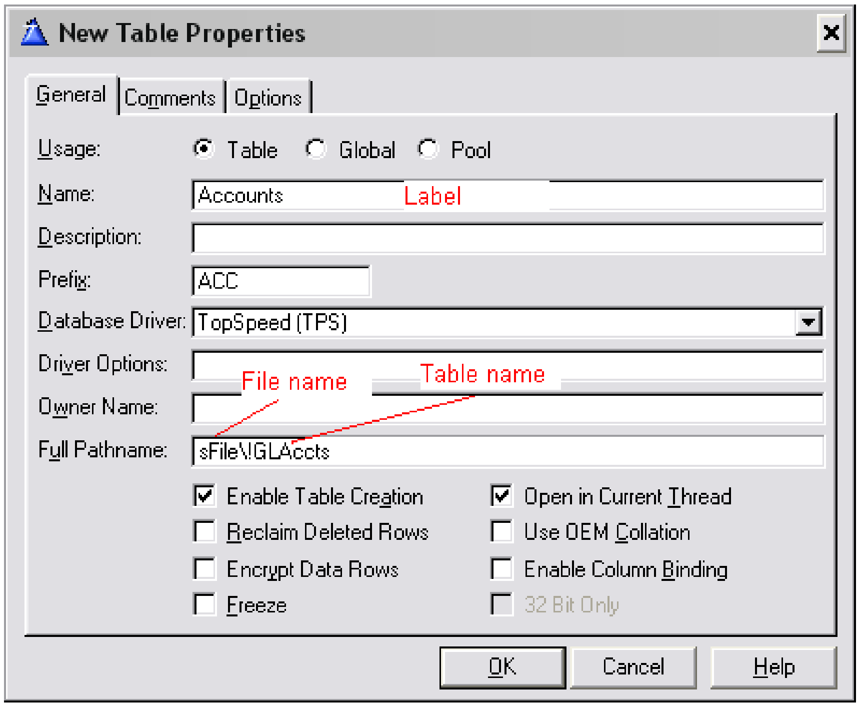
Finally, decide on the "inside name" for each table.

The required syntax is: FileName \! TableName. In SuperFiles.DCT, I have created two files: Accounts and PLAccounts. I decided that I want sFile to appear in Explorer (if that file shows up in my directory, there's no question where it came from) and that, internally, Accounts should be called GLAccts and PLAccounts should be called PLAccts.

Therefore, the Full Pathnames should be:

```
sFile\!GLAccts
sFile\!PLAccts
```

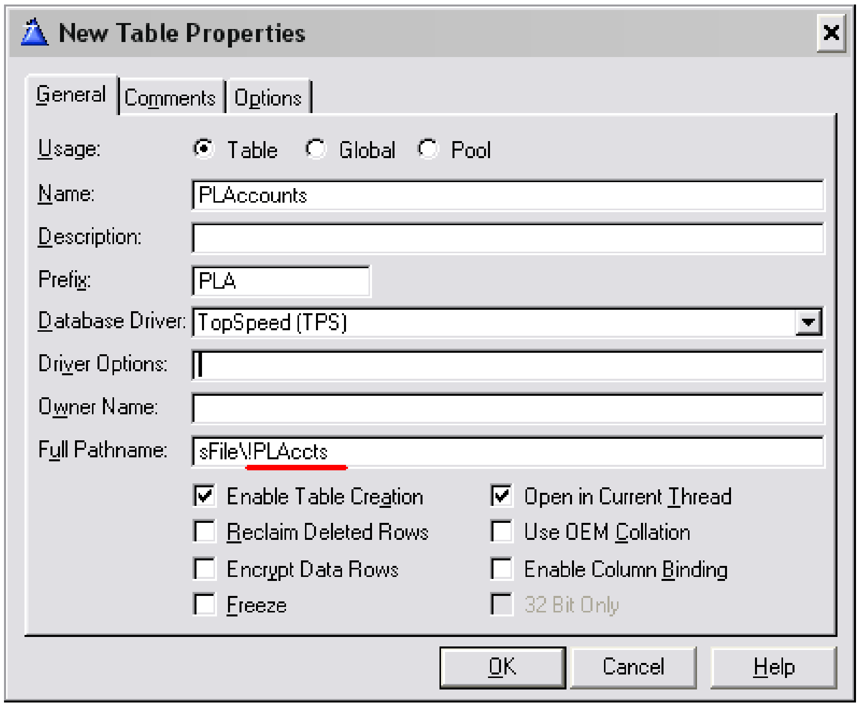
And the final Dictionary configuration should look like Figures 1 and 2.



Usage: [✓ Table] Global Pool
Name: Accounts
Description:
Prefix: ACC
Database Driver: TopSpeed (TPS)
Driver Options:
Owner Name:
Full Pathname: sFile\!GLAccts

- [x] Enable Table Creation
- [ ] Reclaim Deleted Rows
- [ ] Encrypt Data Rows
- [ ] Freeze
- [x] Open in Current Thread
- [ ] Use OEM Collation
- [ ] Enable Column Binding
- [ ] 32 Bit Only

Figure 1. Dictionary spec for Accounts.



Usage: [✓ Table] Global Pool
Name: PLAccounts
Description:
Prefix: PLA
Database Driver: TopSpeed (TPS)
Driver Options:
Owner Name:
Full Pathname: sFile\!PLAccts

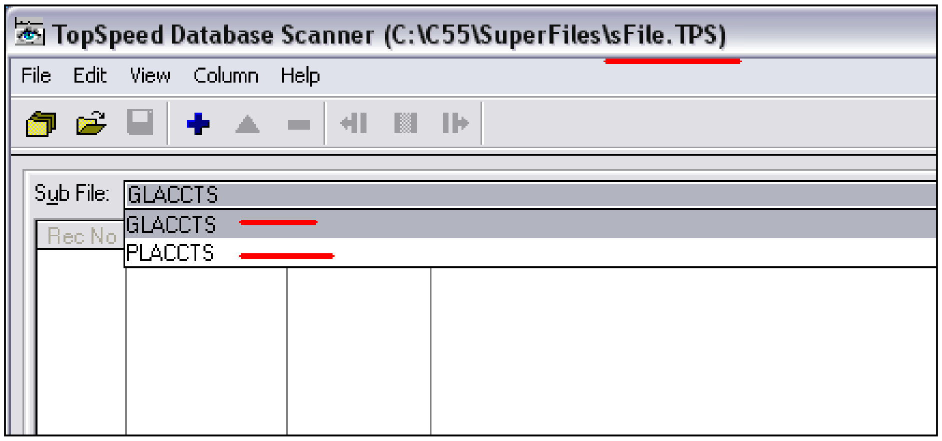
- [✓] Enable Table Creation
- [ ] Reclaim Deleted Rows
- [ ] Encrypt Data Rows
- [ ] Freeze
- [✓] Open in Current Thread
- [ ] Use OEM Collation
- [ ] Enable Column Binding
- [ ] 32 Bit Only

Figure 2. Dictionary spec for PLAccounts

Remember: *FileName* + *special escape sequence* + *table name*. These are all, in a very real sense, irrelevant as far as my application is concerned. The Labels created in the dictionary are still how I code. sFiles is what is created on disk (I don't really care about that). GLAccts and PLAccts are used by the run time library (and I don't really care about that either).

Compile and run SuperFiles.APP or run SuperFiles.EXE (compiled locally, so you don't need any runtime DLLs). Open each browse (you don't have to enter any records if you don't want to). Check the directory and you should see sFiles.TPS.

If you open sFiles.TPS in TopScan, you will see both tables:



Sub File: GLACCTS
GLACCTS
PLACCTS

Figure 3. TopScan shows expected file name and both tables

If you examine the APP file, you will see that there is no embedded code. Anywhere. Creation of the physical file and the internal tables is entirely controlled by the dictionary and the Clarion run time library.

Does Name work for superfiles?

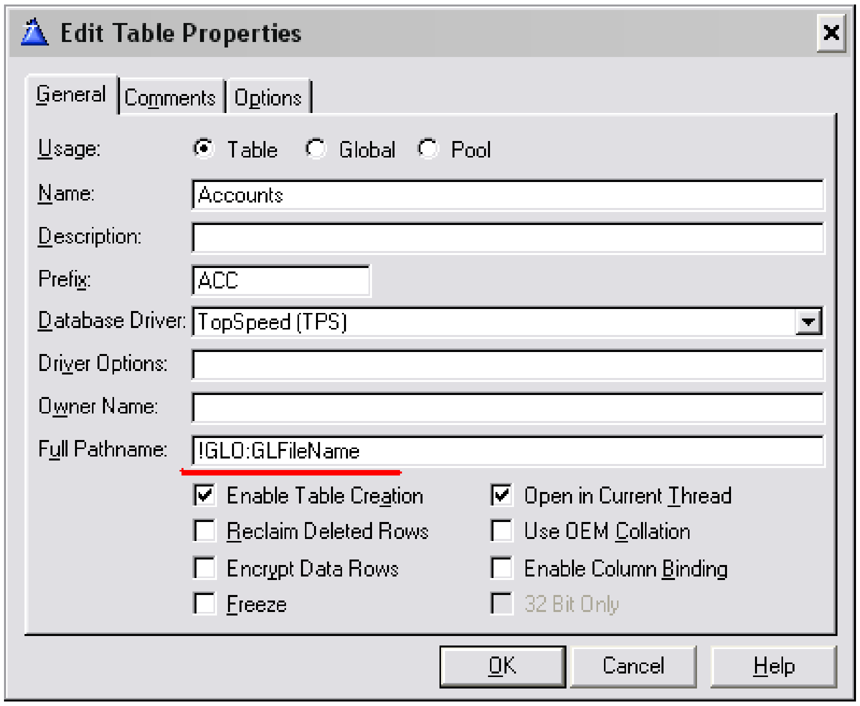
Suppose my app involves multiple companies. The account numbers, staying with the G/L and P/L account tables I started with above, might be different. My code, however, should not change. In this case, I might want a superfile with a different external name - one physical file for each company.

> **Note:** The demo app for this section of the article is superfiles_1.app and superfiles_1.dct.

This is just what using a variable in the Name attribute does with normal files. Will it do the same for superfiles?

Well, the Name attribute takes a string parameter. With superfiles, typically this is taken to mean a string literal, as in the first demo app and the documentation, or a string variable.

I add a pair of global variables, one for each of the two files, to the dictionary and put the variable in the Full Pathname:



Usage: [✓ Table] Global Pool
Name: Accounts
Description:
Prefix: ACC
Database Driver: TopSpeed (TPS)
Driver Options:
Owner Name:
Full Pathname: !GLO:GLFileName

[✓] Enable Table Creation
[ ] Reclaim Deleted Rows
[ ] Encrypt Data Rows
[ ] Freeze
[✓] Open in Current Thread
[ ] Use OEM Collation
[ ] Enable Column Binding
[ ] 32 Bit Only

Figure 4. Nominating a variable for the superfile

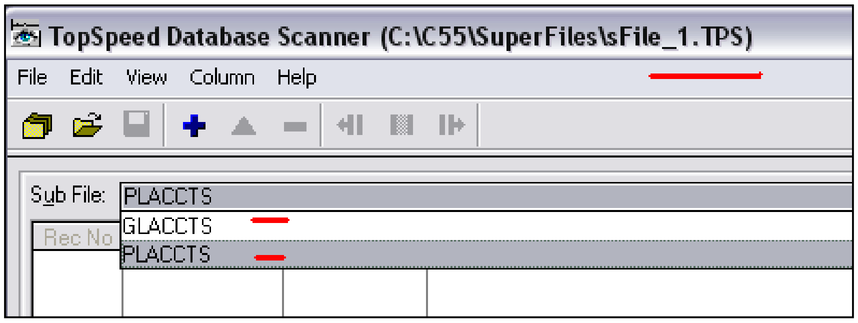
The only thing that is necessary is to prime the variables before any attempt to open the internal tables:

```
GLO:GLFileName = 'sFile_1\!GLAccts'
GLO:PLFileName = 'sFile_1\!PLAccts'
```

Failure to do so guarantees an ErrorCode 45. Conversely, whenever you get an ErrorCode 45, you certainly have not primed a Name variable.

Note that I changed the external file name. This ensures that I can see the file being created.

Compile and run (or run the included superfiles_1.exe) and, lo and behold, sFile_1.TPS appears. And it contains the expected tables:



PLACCTS
GLACCTS
PLACCTS

Figure 5. Name variable creates the expected file and tables

The key, per the docs, is to ensure that the variables contain the magic syntax of *FileName* + *special escape sequence* + *table name*.

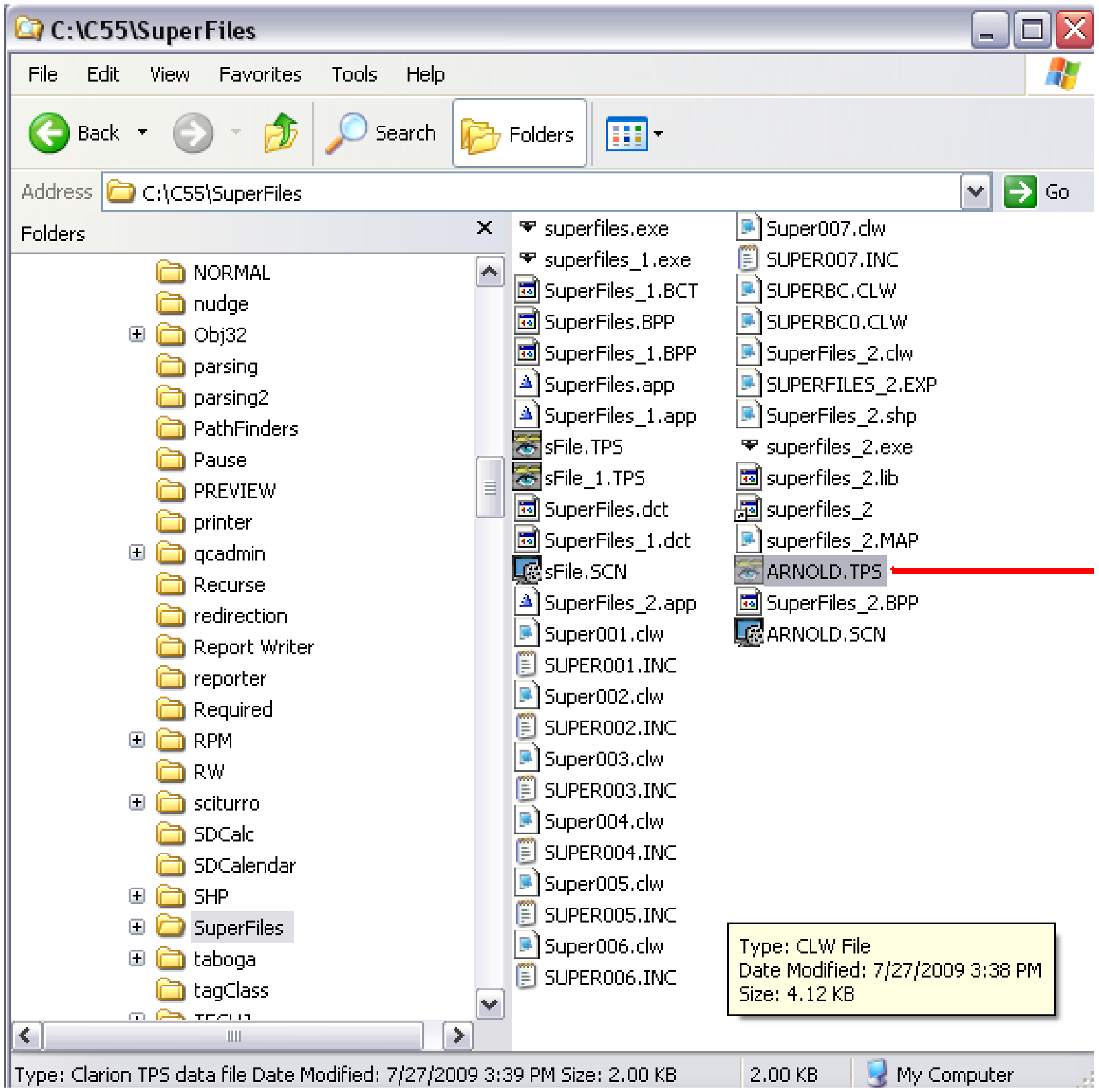
But wait! There's more

If Name requires a string or string variable and I can create a superfile with the expected file and tables name by assigning a properly formatted string to a variable, can I also assign the file name part to another variable? Can I assign the table name part to another variable?

In superfiles_2.APP, I declare a global variable GLO:Nomen (or I could have put it in the dictionary). I give it a value and use it in the Name variable for the superfile:

```
GLO:Nomen = 'ARNOLD'

GLO:GLFileName = Clip(GLO:Nomen) & '\!GLAccts'
GLO:PLFileName = Clip(GLO:Nomen) & '\!PLAccts'
```

and, on running the app, ARNOLD.TPS appears:



C:\C55\SuperFiles

NORMAL
nudge
Obj32
parsing
parsing2
PathFinders
Pause
PREVIEW
printer
qcadmin
Recurse
redirection
Report Writer
reporter
Required
RPM
RW
sciturro
SDCalc
SDCalendar
SHP
SuperFiles
taboga
tagClass

superfiles.exe
superfiles_1.exe
SuperFiles_1.BCT
SuperFiles.BPP
SuperFiles_1.BPP
SuperFiles.app
SuperFiles_1.app
sFile.TPS
sFile_1.TPS
SuperFiles.dct
SuperFiles_1.dct
sFile.SCN
SuperFiles_2.app
Super001.clw
SUPER001.INC
Super002.clw
SUPER002.INC
Super003.clw
SUPER003.INC
Super004.clw
SUPER004.INC
Super005.clw
SUPER005.INC
Super006.clw
SUPER006.INC
Super007.clw
SUPER007.INC
SUPERBC.CLW
SUPERBC0.CLW
SuperFiles_2.clw
SUPERFILES_2.EXP
SuperFiles_2.shp
superfiles_2.exe
superfiles_2.lib
superfiles_2
superfiles_2.MAP
ARNOLD.TPS
SuperFiles_2.BPP
ARNOLD.SCN

Type: CLW File
Date Modified: 7/27/2009 3:38 PM
Size: 4.12 KB

Type: Clarion TPS data file Date Modified: 7/27/2009 3:39 PM Size: 2.00 KB | 2.00 KB | My Computer

Figure 6. Variable for file portion creates the correct file

I've included the resulting files, so you can use TopScan to verify that the correct table names are created.

Finally, in superfiles_3.app, I create an additional variable for the table part:

```
GLO:Nomen        STRING(20)
GLO:Nombre       STRING(20)
```

All variables are assigned before forming the Name variable:

```
GLO:Nomen = 'MILTON'


GLO:Nombre = 'Filbert'
GLO:GLFileName = Clip(GLO:Nomen) & '\!' & GLO:Nombre


GLO:Nombre = 'Cashew'
GLO:PLFileName = Clip(GLO:Nomen) & '\!' & GLO:Nombre
```

Note that I have, again, changed the external file name.

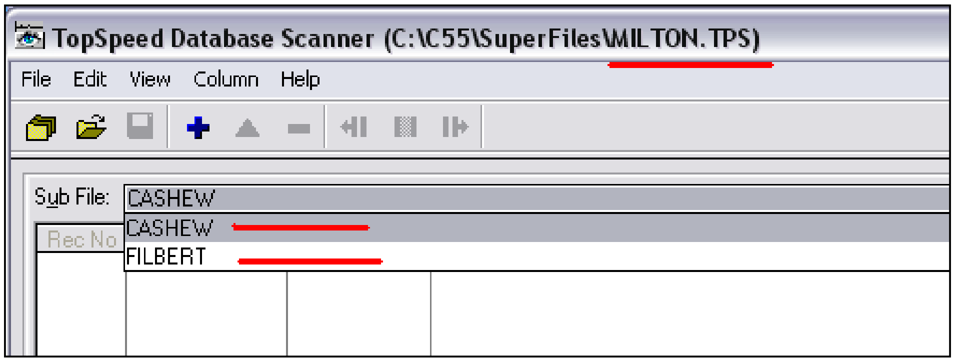

TopSpeed Database Scanner (C:\C55\SuperFiles\MILTON.TPS)
File Edit View Column Help
Sub File: CASHEW
Rec No
CASHEW
FILBERT

Figure 7. Correct file and table name created

Et voilá, everything precisely as expected.

Summary

In these two articles, I've demonstrated that the structure of a superfile is more correctly thought of in the same terms I use in thinking and talking about SQL files and tables. I've also demonstrated that all TPS files are inherently superfiles.

What distinguishes a single-table TPS file from a multi-table TPS file is the syntax of its Name. And the only unique requirement of that syntax is the \! delimiter. Whatever is to the left is the name of the physical file. Whatever is to the right is the (internal) table name.

Other than that, superfiles are amenable to all the standard programmatic technique I use for any "flat" file.

The source code zip contains C5.5 versions of the sample apps (which can also be opened in C6) as well as converted C7 versions.

Download the source

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

Reader Comments

Posted on Monday, September 14, 2009 by Don DeRespinis

GLO:GLFileName = Clip(GLO:Nomen) &p '!' & GLO:Nombre
The "p" seems to be a typo

Posted on Monday, September 14, 2009 by Dave Harms

Thanks Don - fixed.

Dave

Posted on Friday, September 18, 2009 by Michael Gorman

Steve:

Thanks for the article....

Suppose I have an existing database with a whole bunch of TPS file and I want to convert them to be inside a super file?

Is there a magic bullet or do I make a SuperDestination file and then write a series of one-table conversions to boot strap the SuperDestination into existance?

Thanks in advance,
MikeG

Posted on Friday, September 18, 2009 by Steven Parker

Mike,

How's 'bout after new years (i.e., next week, week after) I do that article?

Posted on Tuesday, September 22, 2009 by Michael Gorman

Steve:

Knowing you I know which New Year's to calibrate to....

Thanks,
Mike G

Add a comment

Clarion Magazine

Clarion 8 Wish List: Overview and Comments

by Dave Harms

Published 2009-09-17

Earlier in the summer Robert Hutchison suggested I canvas Clarion developers for their wants and wishes for Clarion 8. I've organized those suggestions and highlighted the key points below.

In the main the comments were supportive of SV focusing at least some of its energy on the Win32 development path, but there were some dissenters. I've separated out those comments and have added my own opinions as well.

In the following list my comments are inlined, in italics. In all cases the emphasis added is mine.

- Robert Hutchison: Have the **Window designer and the embed tree open at the same time** and be able to edit them both at the same time.
- Robert Hutchison: A **multi-tabbed embed tree** in the same manner as they have done multi-tabs in the dictionary editor. Every time you open an embed point from within the designer, say from a control, it would open another embed tab within the Primary embed tab. Lee White points out that it would be nice to have different keystrokes, perhaps Enter for one tab and Ctrl-Enter for multiple tabs.
- Ben Dell: **Enhanced team support**. Ben also mentioned the ability to **snapshot a project** and save all the source, tmp, inc, clw, trn and all other relevant files. That isn't necessarily a team feature, but you could use this to help manage a team project if you could distribute that snapshot to team members. *Actually this is really two issues. One is the ability to duplicate the project, and the other is the ability to duplicate the development environment, in particular the templates and third party products.)*
- George Lehmann: Whether you believe in C7 or .Net as the wave of the future, I think **team development tools** is the

place where the product family could use the most help. I know various attempts have been made in the past to integrate different source control systems into the Clarion world, but I've yet to see anything that comes close to the functionality of what can be achieved in the pure source world.

Yes, the binary structure of the app files is the reason why our world is so different from everyone else's. But until a **decent source control system** is built into Clarion and not added on as an import/export afterthought, we'll never have productive team development.

- Ben Dell: The ability to **prototype screens and the dictionary at the same time**. *(It turns out that the ability to update the dictionary from within the AppGen, in C6, isn't something SV would like you to do. It's viewed as a bug, but has become an important feature for some developers.)*
- Ben Dell: **Extract screens from an APP and display them in a viewer** for client approval/commentary.
- Michael Summons: Using Data Modellor we've always had the ability to **edit the dictionary while the app was open**. I would expect to be able to have at least the same functionality with C7. It is a feature I use a lot. *(Data Modeller is not included in Clarion 7 - in part this functionality has been replaced by the Dictionary Diagrammer, but the DD is a display only feature right now. That is, you can't use it to update the dictionary.)*
- Stephen Bottomley: **Virtual Dictionaries made from tables in multiple dictionaries**. *(Steve called this a "perennial favorite" and he's right, the ability to use more than one dictionary has been requested many times over the years. It really would be useful for the massive applications many Clarion developers create.)*
- Thomas Glomb: **64bit compiler please!**
- Kevin Plummer: **64bit compiler please!** I'm paying for C7 (and C8) but have not used it yet. My initial enthusiasm when I signed up has dispersed with the long release delays. I'll get around to it when I have time but a 64 bit compiler will make me get my feet wet quicker...
- Bruce Johnson: **A 64 bit compiler** would give existing apps a huge boost in terms of longevity without the economic pain of a rewrite to .NET. I agree with other posters though that beyond that, SV are probably better off spending their time on . Net, once C7 is complete. *(I think Bruce has, as usual, nailed it. Although I'm not seeing a huge demand for native 64 bit Clarion applications yet, it's growing, and being able to create native 64 bit apps is increasingly becoming a selling point. For some Clarion developers having a 64 bit compiler will be an important hedge against having to move a massive code base to .NET.)*
- Paul Blais: An **improved resources files editor**. Maybe even have an external table like the DCT that can hold all

these common things developers often reuse app to app. You can add images, icons, and other resources on the solution but when you need you pick one you can't ever use it to pick you go to the disk all the time for every item in every instance. The icon picker is lame. No one really uses the ones that are easy to pick from the drop down list and no drop down can really hold all of them. It would be nice to make currently added images, icons, and any other resources (tabbed by type) in a browse. You then can pick from the ones you have or add a new one and pick it. This might lead to a better solution explorer too. When building an app I should interact with all the baggage I'm adding as I go and never have to reselect things I already have selected.

- Casey Rippon: I am more interested in improvements to my users' experience than minor efficiencies in the IDE. A couple of much needed improvements: 1) **Browse cells with word wrap like HTML tables**, 2) **Multiple select that uses Windows standard click, Ctrl-Click and Shift-Click**. *(Multi-select, or tagging, has long been the domain of third party vendors. I don't actually know if any of them provide a tagging solution that mimics list box multi-select, but it can't be that hard to do.)*
- David Sperling: As more and more of my customers seem to have at least one Mac, it would be great to be able to compile **native Mac version of apps** as well. *(A native Mac version of the Clarion compiler, RTL and file drivers seems like a monumental task to me. On the other hand, if you're doing .NET development then you should be thinking about Mono on the Mac. Have a look at Stack Overflow for recent discussions of this topic.*
- Murray Gillespie: **Embeds, Extensions, Calls and formulas should have their own pads**. (as in data, code templates etc). In C7 getting to the embed editor especially is a regression in terms of usability from C6.
- David King: I'd like to see **a template that does the two browse window** feature, like Invoice .. Detail with a wizard. And following the thread, I agree with the thought that the .Net path is where the future of Clarion should be. Including mobile app gen.

Objections

The following commenters disagreed with any attempt to continue developing Clarion (Win32) beyond version 7.

- Martin Howes: Talk of Clarion 8 is a distraction. The SoftVelocity development team should be **spending their time on . Net because that is the platform for the future**.

- TrevorC: First we should not even be discussing V8, when V7 is still not yet fully-feature complete (imo) compared to other modern development platforms. It may be feature-complete as per the original roadmap, but that is not my point. Second I agree with another poster about .NET... I can't see how a company with the limited resources of SV can possibly develop (with any professionalism) TWO platforms (c7 and C.net) - they should choose ONE and go for it! I believe long-term **they should choose Clarion.Net and put ALL their efforts after C7.x into developing ONLY Clarion.Net and focus in particular on ease of migrating apps from C6.x/C7 to this platform**... DotNet opens up MANY more opportunities than C7/C8 ever will. Look at all the major component/add-on writers... all their time & money is in .Net and we need to be using their products natively to keep our apps current/ahead of the game. We've battled too long to get Clarion working with these tools in COM/OCX form... Clarion needs to get "mainstream" or risk dying. Furthermore think about it SV, put your marketing heads on for a minute! **Clarion.Net is the ONLY area (RAD developing & template code generation for .Net) where SV truly have a unique-selling-point over other platforms**... so USE that advantage before we all run out of money & patience! Sorry, long answer... the short answer is we shouldn't be discussing C8 imo.
- Michael Gould: I'm not interested at this time with a C8. It would be nice if ActiveX's were as easy to use in C7 as they are in other development platforms, but for me personally C7 will be the last Win32 platform that I will use. SV should concentrate on fixing those items that are specific to C7. We all know that Clarion 7.1 is going to see the light of day because that is where the new report writer is promised, but in 2009 IMO opinion **SV should be concentrating their efforts on Clarion#.**
- Luuk Sluyter: I agree **all efforts should be put into the Dot.net path**. We have been (mis)using Clarion from version 2 onwards to generate C, VB and VB.Net code for pc and mobile devices, making extensive use of dictionary and home made templates. Today Windows CE5 and WM form the majority of the mobile devices we sell and most projects start in Clarion 6.3 when we create a Vb.Net application using the templates. At a certain point the application is frozen and manually coded further using VS2005. Of course we would love to stay within Clarion# with support for VB.Net and C# so we can stay within the one environment!

The whole Win32 vs .NET topic is one that has a lot of potential to divide Clarion developers. Some of us have massive Win32 code bases that may not port all that easily to .NET, and don't face compelling market pressures to create .NET versions of their applications. For these folks, a 64 bit compiler and IDE improvements will suffice for years to come.

For other developers who do feel the push to create .NET versions of their applications, as indicated in the last three comments, energy put toward the Win32 product detracts from the all-important .NET version of Clarion.

Even the most die-hard Win32 Clarion developers will, I think, agree that the future of Clarion is .NET. It's unrealistic to think that Clarion 7 and its successors are going to have wide appeal outside the existing Clarion community. There just aren't a lot of devs out there looking for new Win32 tools.

The real question, then, is what kinds of applications will the Clarion.NET AppGen generate? When it comes to desktop applications there are, broadly speaking, two main alternatives available to SoftVelocity (and they aren't mutually exclusive). One is to create backward-compatible applications using the Clarion# language. Think of this as an ABC# template set.

Option one: Clarion# ABC#

No, SoftVelocity hasn't said anything about a product called ABC#, at least not that I've heard. This name is entirely my own creation. Take that for what it's worth.

The upside to an ABC# template set is easier porting of existing applications to .NET. There are huge differences in how windows are handled in .NET (the WINDOW structure has been replaced by WinForms classes), and some other significant changes to the Clarion# language. But on the whole SV has done quite a good job of maintaining backward compatibility, particularly when it comes to data access and business logic.

The downside to an ABC# template set is the ABC architecture, like the Clarion (legacy) architecture, is a bit long in the tooth. Modern application architectures separate the user interface, business logic and database access into more or less separate layers. Clarion applications tend to lump logic, data and UI together into single procedures, which makes maintenance and testing more difficult.

ABC# could benefit a lot of current Clarion Win32 developers. But it's not going to have any appeal outside the Clarion community. Let's face it: the rest of the .NET world uses primarily VB.NET and/or C#, and a twenty-year old application architecture is going to be just as tough a sell as a proprietary language.

Option two: C#/VB.NET multi-tier

Yes, you can generate C# and/or VB.NET code in Clarion. For certain kinds of applications where you don't depend on the form designer for the final appearance (such as MVC web apps and, potentially, WPF apps), even C7 is actually a pretty good fit.

Clarion-style end-to-end code generation is, to the best of my knowledge, absent in the world of .NET development. Oh, there are some pretty good code generators out there. But none of them abstract the underlying architecture like Clarion does. None of them give you anywhere near the same power that's in the Clarion template language.

The .NET world *needs* Clarion's code generation technology. They just don't need ABC or Clarion#, as much as they may matter to Clarion devs.

At some point this kind of AppGen really needs to be a VS plugin as well. SoftVelocity has indicated that's something they'll consider.

Okay, there's one more option, I suppose.

Option three: Clarion# multi-tier

Theoretically a Clarion# multi-tier template set is an option, but I don't think it should be a first choice. In order to implement a template set you generally need a working source example (at the least, that's going to make your life a lot easier). And there are lots of examples out there of best-practices .NET code, written in VB.NET and C#.

There are only two practical ways I can think of to create a quality Clarion# multi-tier template set. One is to port the examples to Clarion# and then write the templates, and the other is to create the templates for C#/VB.NET and then port the templates to Clarion#. I'm not sure one is more work than the other, but certainly the second option results in *two* template sets, not just one.

And I think it would take less time to achieve that first working C#/VB.NET template set because Microsoft's compilers are industry standards. There could well be enhancements needed to the Clarion# compiler, which unavoidably lags Microsoft's compilers.

Which is better?

Which option is better, an ABC# template chain or a C#/VB.NET multi-tier template chain? Until such time as there's a VS plugin version of Clarion.NET, probably the former. SV has done a lot of work to maintain backward compatibility with the Clarion language, and that means there is a path for preserving the value of all that business logic you've written over the years.

But whichever option appears, or whichever path you take to .NET (or even if you stay in Win32), there are ways to make your code more maintainable and portable. I'll have more to say on that subject in the weeks and months to come: just keep your eyes open for a new series in the mag on *refactoring embed code*.

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

Reader Comments

Posted on Friday, September 18, 2009 by Gerhard de Jager

I HAVE TO AGREE WITH "TrevorC":

"DotNet opens up MANY more opportunities than C7/C8 ever will. Look at all the major component/add-on writers... all their time & money is in .Net and we need to be using their products natively to keep our apps current/ahead of the game.... Clarion needs to get "mainstream" or risk dying. Furthermore think about it SV, put your marketing hats on for a minute! Clarion.Net is the ONLY area (RAD developing & template code generation for .Net) where SV truly have a unique-selling-point over other platforms..."

I STARTED WORKING A YEAR AGO WITH CLARION.NET - PREVIOUSLY A CLARION WIN32 PROGRAMMER EVEN IN THE DOS YEARS CLARION WAS THE BEST (RAD TECHNOLOGY) AND THE CLARION DOTNET VERSION WILL DEFINITLY BE MORE ACCEPTED IN FUTURE. I AM SURE THAT THE APP. DESIGNER WILL BE AS IMPRESSIVE AS C7 (HOPE TO SEE IT SOON) AND THAT SV SHOULD LOOK AT THE BIGGER PICTURE,THEY HOLD THE KEY TO THE FUTURE OF MORE INTELLIGENT RAD DOTNET SYSTEMS EVEN COMPARING IT TO MICROSOFT VS 2008 THEY COULD TAKE THE MARKET.

Posted on Monday, September 21, 2009 by Rhys Daniell

Let's try some perspective here. CPD hooked a lot of us back in the DOS days because it was a true paradigm shift compared to the tools available at the time, giving us a tool that was both easy to learn and use and at the same time could be extended almost indefinitely by a developer with complex needs.

Windows database applications aren't going away any time soon, the main alternative (.Net) just isn't rich enough or responsive enough. However we use C6 to do some pretty advanced stuff (including multi-tier) and I've yet to come across something that couldn't be easily addressed by a template tweak, a third party tool, or some clever hand code. I just hope that creaky old IDE can stagger along until we've got time to move to C7! So there we have a mature market and a mature tool which is still the best in that business.

When it comes to .Net we're just playing catchup, and far too slowly. By the time Clarion.Net is mature, .Net will be hopelessly obsolete. Take the blinkers off, guys; it's a horrible, bloated, over-complex environment for what *we* do, i.e. databased apps. Given the choice between a .Net app and a windows thin client, our users will take the thin client every time.

An alternative to .Net *must* emerge, and that's where SV's meagre development resources must be aimed. It's time to leap ahead of the current status quo. Clarionet *should* have been the answer but nobody had the time or inclination to develop or market it properly. I don't know what the Next Big Thing in database development will be, but I bet somebody out there knows. SV doesn't have the muscle to take on the majors, so it has to outmanoeuvre them.

What we need is the kind of thinking that went into CPD, applied to a clean sheet of paper. Anything else is just a 'make work' project.

Posted on Tuesday, September 22, 2009 by Dave Harms

Rhys,

Interesting perspective. While I agree CPD was a creative new solution, I have to take issue with "could be extended almost indefinitely." I think that better applies to the Windows product. The CPD model files were pretty primitive by comparison with the template language, and if you really wanted to extend a CPD app you needed one or more LEMs.

"it's a horrible, bloated, over-complex environment for what *we* do, i.e. databased apps."

heh heh - I seem to recall the same sort of reaction to CDD from some folks<g>.

There's no question at all that I can be more productive right now for line of business apps with C6/7 than with .NET. But IMO that has far more to do with the template technology than with .NET vs WinAPI.

The *opportunity* for bloat and complexity is greater in .NET than ever before. But that's nothing new. It's also greater in WinAPI than in DOS.

The question is how you manage the complexity. In Clarion, the pure template-driven approach reached a point of unmanageability, whereupon much of the logic got offloaded to the ABC class library. I think most Clarion developers these days see ABC as an improvement over Legacy, even though there were many who objected to the increased complexity.

There are ways to manage complexity in .NET that are simply impossible in WinAPI apps.

"An alternative to .Net *must* emerge,"

If you mean the framework itself, then the sheer size, scale, and current rate of adoption argue it'll be around for a long time. WinAPI, which is far less complex, has lasted 25 years and will be here for many more to come. The mono project, about which I've always harbored doubts, actually looks like it's doing a pretty good job of bringing .NET to other platforms, including Linux and the Mac. If anything, .NET is getting stronger.

The surface *developers* use, however, can and will change. There are many different aspects to .NET. If by "an alternative" you mean an alternative to WinForms, or ASP.NET, then not only must an alternative emerge, but it's already happening on both fronts. Eventually ASP.NET MVC and WPF/Silverlight will no doubt be replaced too. But it seems quite unlikely to me that you and I will see .NET pitched in the garbage bin during our working lifetimes.

And if by "alternative" you mean an alternative to transplanting AppGen in its current form into .NET, then yeah, I would agree that in the long term there's opportunity for much greater impact there. But SV needs to take care of its current customer base too. Tricky.

"What we need is the kind of thinking that went into CPD, applied to a clean sheet of paper."

And we need the kind of cash Bruce Barrington was sitting on when he had the luxury of starting from a clean sheet of paper<g>. But really it wasn't a complete tabula rasa. He was trying to solve existing problems, and he brought past experience to bear.

The great problem in the .NET dev world, as I see it, is almost the same problem Barrington addressed in his day, at least with CPD. It wasn't that you couldn't create great applications with the available tools, it was that it was unnecessarily complex to do so. CPD provided a standard yet customizable architecture, and since that day Clarion has continued to refine the concept.

The problem for Clarion, as I see it, is that one of the greatest strengths of the system is also its greatest weakness: the Clarion language. If it wasn't for the language, most of us wouldn't be using Clarion. But continually upgrading the language from DOS to Windows to .NET, and competing with Microsoft's languages, has also taken enormous resources.

Somewhat naively I thought this resource drain would get smaller with Clarion#, since the compiler only has to emit IL code which can easily be verified against, say, the C# compiler's output. But really it's getting larger, since there are so many different ways now to do development in .NET. Do you target WinForms? WPF/ Silverlight? ASP.NET MVC? Mobile?

Meanwhile what the .NET world needs, desperately, is a way to accomplish what Clarion developers have enjoyed since CPD: a customizable application framework where you can be productive from the getgo. And really it needs it in the context of the Visual Studio ecosystem, and generating C#/VB.NET code.

None of that, I suspect, matters much to a fairly large percentage of Clarion developers who will be quite happy to use Clarion Win32 (and hopefully 64) until they reach retirement.

These are interesting times.

Add a comment

Clarion Magazine

Writing Your Own Template Chain: Creating Embeds At Runtime

by Dave Harms

Published 2009-09-21

This is the third article in a series on writing your own template chain. In Part 1 I talked about the possibilities of generating non-Clarion code using the Application Generator and a custom template chain. In Part 2 I explored some of the concepts needed to understand how embeds work. In this article I'll take a more practical approach and show how to create embed points on the fly.

Why flying embeds?

As you probably recall, there are four basic building blocks in the template language: #Procedure, #Code, #Extension, and #Control templates. #Procedure templates model procedures; #Code templates allow you to easily insert a template's output into an embed point of your own choosing; #Extension templates insert code into multiple embed points of the template's choosing; and #Control templates are wrappers around on-screen controls.

#Code, #Extension and #Control templates all act in the context of a #Procedure, which corresponds to a procedure in the AppGen. You put the core of your code generation in the #Procedure template, and then your #Code, #Extension and #Control templates typically generate code into embed points defined in the #Procedure section.

For example, I'm currently working on a template chain that generates C# for web applications. (This is an in-house project, and it's in support of an upcoming new version of the Clarion Magazine web site.) Among other things I have #Procedure templates that generate C# classes, and the #Control templates generate code into embed points in the class.

Now my #Control template needs to add one or more methods to the class. If there's only one of these #Control templates per #Procedure, that's not a problem: the #Control template can simply generate the entire method into an embed point inside the class structure. But what happens if I have two instances of the #Control template? I'll have two identical methods generated into the same class, and the compiler isn't going to like that (and neither will I).

I *could* exercise some control over the generated code via the #Procedure template. One possibility is to actually generate the method (with embed points) in the #Procedure template, but only if there's at least one instance of the #Control template in question. Another is to have some sort of flag in the #Procedure's scope, such that only the first instance of the #Control template creates the method. But both of these approaches are pretty ugly. They require you to maintain code in two places. And they just have a bad code smell.

Here's a rule of thumb: as much as possible, #Control and #Extension templates should not rely on *logic* in the #Procedure template to do their job.

A better approach, one that keeps the logic where it belongs, is to create what I call an *embed loop*.

Embed loops

Embed loops are simply #For statements that result in the creation of potentially multiple embed points, each of which can also contain multiple embed points (and so on).

Here's what the code looks like:

```
#For(%LoopEmbedSymbol)
  #Embed(%LoopEmbed),%LoopEmbedSymbol
#EndFor
```

%LoopEmbedSymbol is simply a multi-valued symbol of my own choosing - it could be any multi-valued symbol (and it can have dependent symbols as well).

The #For loop will execute the #Embed statement once for each instance of the symbol. And unlike the #Embed statements I showed in Part 1, this statement has an extra parameter on the end, which happens to be my loop symbol.

The key to #Embeds

This extra symbol tacked onto the #Embed statement is the key to some really awesome code generation. This statement:

```
 #Embed(%LoopEmbed),%LoopEmbedSymbol
```

can be translated as "create an embed point called %LoopEmbed for every instance of %LoopSymbol." And having those embed points, you can now generate code into them using a matching version of the #At statement:

```
#At(%LoopEmbed,'Some string value')
 ! generate this code
#EndAt
```

Notice the second parameter of the #At statement. As long as this value matches an entry in %LoopEmbedSymbol you'll get code generated at the embed point.

Of course, in order to make all of this happen you need that multi-valued symbol. There are several ways to declare that multi-valued symbol, including the following:

1. #Declare at the procedure level
2. #Declare at the global level
3. #Prompt at the procedure level
4. #Prompt at the global level

I'll look at each of these in turn.

#Declare in #Procedure

If you're like me you'll start off by trying to #Declare the embed loop symbol at the #Procedure level. This makes perfect sense, since I'm using the symbol only within this procedure.

```
#Declare(%LoopSymbol),Multi
```

But there's a problem. The place to #Declare a #Procedure symbol is just after the template prompts. In my experience symbols created this way aren't in scope for #Extensions. So scratch that option.

#Declare in #Application

An alternative is to declare the symbol globally, but that's going to raise some issues too. What if other procedures use the same extension template? They'll add records to the symbol which will result in spurious code generation. You can avoid that by issuing a #Free statement on the symbol in the #AtStart section of the #Procedure:

```
#AtStart
  #Free(%LoopSymbol)
#EndAt
```

But make sure you do this in the #AtStart section - do it later, once code generation has started, and you'll wipe out any data your templates have created.

Using #Prompt

Another option, and the one I tend to prefer, is to use a #Prompt at the #Procedure level (or the #Application level, if the circumstances warrant):

```
#Prompt('Embed loop symbols',@s200),%EmbedLoopSymbol,Multi('')
```

One important difference between #Prompt and #Declare is that #Prompt values are persisted. (#Application-level #Declare statements can also be persisted if the have the ,Save attribute, although I haven't had much success with this approach.) Another is that #Prompts are by default visible, and you may not want the template user to either be distracted by such prompts or inadvertently change some data. Fortunately, it's easy to hide prompts:

```
#Boxed('Hidden stuff'),Where(%False),Hide
  #Prompt('Embed loop symbols',@s200),%EmbedLoopSymbol,Multi('')
#EndBoxed
```

Presto! No more prompt. This is a most useful technique - I now use it exclusively for persisted symbol values both at the procedure and at the global level. And if you change %False to %True you can view the data, which can come in handy when debugging.

Handling duplicates

In all of these examples I've used the ,Multi attribute to declare a multi-valued symbol. But ,Multi allows for duplicates, so in most cases you'll want to use ,Unique along with ,Multi to guard against accidentally creating duplicate embeds. (If you #Declare the variable instead of using a #Prompt you can use ,Unique *instead of* ,Multi.)

A routine example

I'll come back to my C# example in the next installment, but first here's a less complicated example that uses more familiar Clarion terminology.

I've updated my #Procedure template with an embed loop that provides to the ability to generate one Routine for every value in the symbol %Routine:

```
#Procedure(Generic,'A generic procedure'),WINDOW
#Sheet,HSCROLL,ADJUST
  #Tab('&General')
    #Display('Template prompts go here')
    #Boxed('Hidden stuff'),Where(%False),Hide
      #Prompt('Routines',@s200),%Routine,Multi(''),Unique
    #EndBoxed
  #EndTab
#EndSheet

! Routines

#For(%Routine)
%Routine Routine
 #Embed(%RoutineEmbed),%Routine

#EndFor
```

I've also created a new extension template:

```
#Extension(EmbedLoopDemo,'Demonstrate an embed loop'),Procedure,Multi
#Display('No prompts for this extension')
#AtStart
  #Add(%Routine,'EmbedLoopDemoRoutine' & %ActiveTemplateInstance)
  #Insert(%Debug,'EmbedLoopDemo added Method')
#EndAt
#At(%RoutineEmbed,'EmbedLoopDemoRoutine' & %ActiveTemplateInstance)
  ! Routine code added by extension instance %ActiveTemplateInstance
#EndAt
```

I create a new APP using my custom template chain (see the downloadable zip for an example) and I create a new procedure. Since at present there's only one #Procedure in the chain I don't have to make any choices as to procedure type. Then I add two instances of my EmbedLoopDemo extension template to the procedure (Figure 1.)

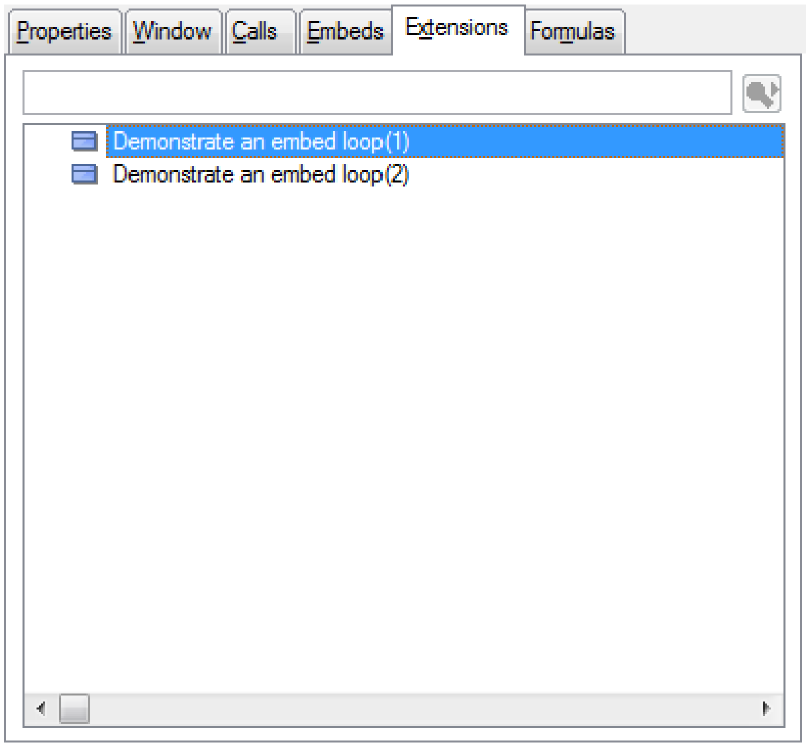


Figure 1. Adding two instances of the template

Here's the resulting generated code:

```
! Routines

EmbedLoopDemoRoutine1 Routine
   ! Routine code added by extension instance 1

EmbedLoopDemoRoutine2 Routine
   ! Routine code added by extension instance 2
```

That's fine, but so far these extension templates are generating code for their own routines only, so there's no great advantage of just having a single embed point. So here's a second extension template (line break added):

```
#Extension(EmbedLoopDemoAddition,'Add extra code to the embed loop ↵
  demo extension'),Procedure,Multi
#Prompt('Choose an routine',From(%Routine)),%ExtensionTarget
#AtStart
#EndAt
#At(%RoutineEmbed,%ExtensionTarget)
```

```
    ! Routine code added by the second extension template
#EndAt
```

This template adds some additional code to a routine, but which routine is determined by the template prompt, as shown in Figure 2.

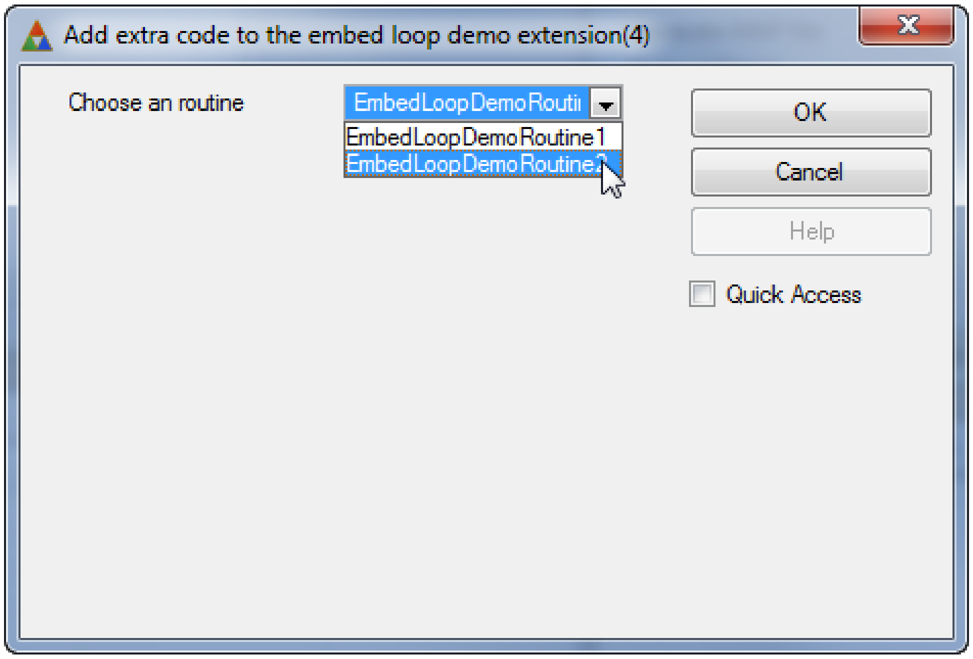


Figure 2. Adding the EmbedLoopDemoAddition template

Now the output is as follows:

```
! Routines

EmbedLoopDemoRoutine1 Routine
   ! Routine code added by extension instance 1

EmbedLoopDemoRoutine2 Routine
   ! Routine code added by extension instance 2
   ! Routine code added by the second extension template
```

Notice what's happened. The EmbedLoopDemoAddition template is now generating code *into an embed created dynamically by another template*. Think about that. The name of the embed (%RoutineEmbed) is fixed - you have to know that when you write your templates. But the contents of any symbols used in embed loops (and you can have more than one) are completely dynamic.

All you need to know to generate code into a particular point in an embed loop is the value(s) of the appropriate symbol (s). That's an immensely powerful feature.

Next time I'll show a more complex example that generates some of that C# code I alluded to earlier in the article.

The source code is in the form of a C7 solution containing a project for the templates and another for the example app. It's much, much easier to work on templates in C7 than it is in C6. To create the application in C6 just register the template chain (you'll need to copy the template files into your template directory or update your redirection file to point to the templates' location), create a new application with a single procedure, and add the extension templates to the procedure.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Template Logging/Debugging Revisited

by Dave Harms

Published 2009-09-25

About three years ago Clarion Magazine published an article titled A Template Debugger. In that article Russ Eggen described Mark Goldberg's technique for logging messages from the template language to the system log, so those messages could be viewed with the freely available DebugView utility.

I was reminded of this article recently when Roelf Du Preez posted a question in the C7 newsgroups about calling DLLs from templates, and Russ posted the link.

Up until that point I hadn't tried the template debugger technique in C7, mainly because template #Message statements are already much easier to view than in C6. They get logged to the output window which you can drag out of the IDE and resize as you like. But unlike DebugView, the output window doesn't offer any filtering, color coding, searching or auto-scroll disabling, all of which are very handy when you're looking at a lot of debug statements. So I was inspired to try the template debugger in C7.

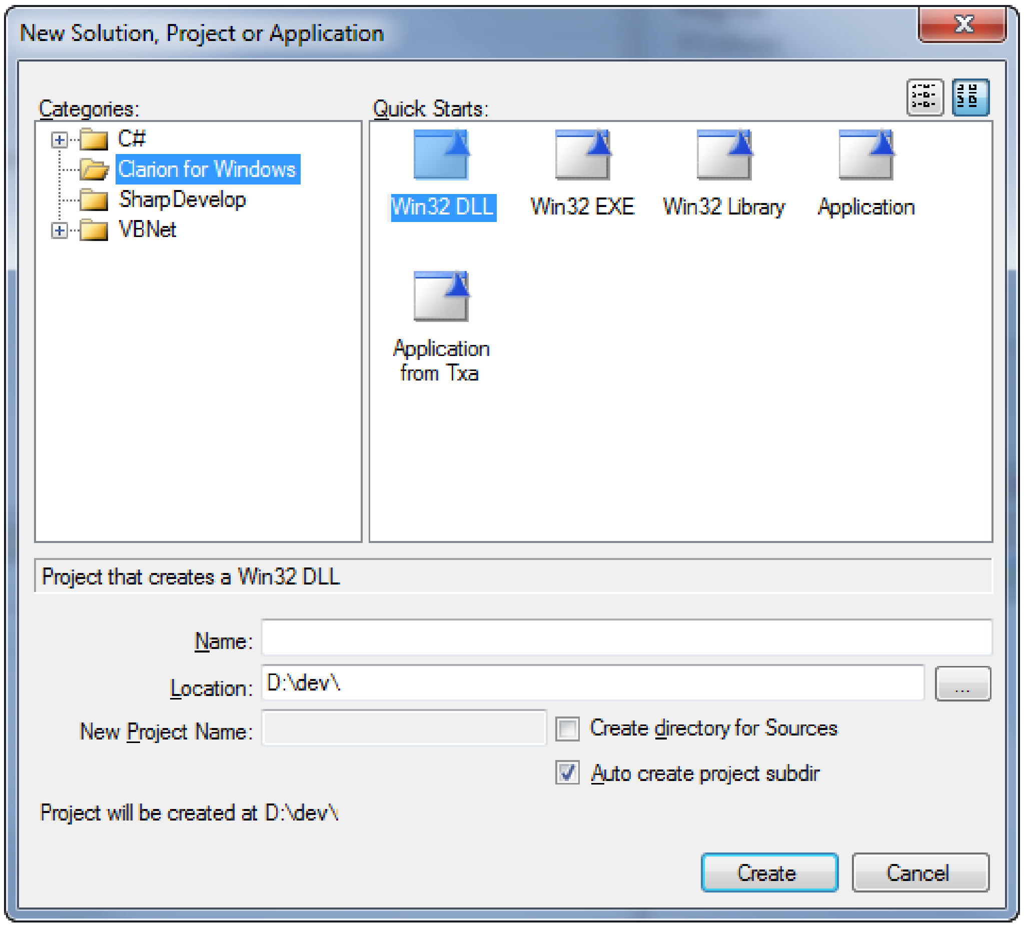
> **Note:** Okay, technically this really isn't a template debugger, since it doesn't let you set breakpoints or watch variables. It's a logger. But it's still extremely useful tool.

The C7 version

I won't go into all the gory details of how the template debugger works; if you haven't tried it, you should first have a look at Russ's article to get the general drift.

Here's how to create a C7 version.

You can either create a new hand coded project, or use the one in the downloadable source at the end of this article. To create the project from scratch choose File | New | Solution, Project or Application. Select Clarion for Windows and Win32 DLL, as in Figure 1. Give the project a suitable name.



Project that creates a Win32 DLL

Name:

Location: D:\dev\

New Project Name:

Create directory for Sources: ☐

Auto create project subdir: ☑

Project will be created at D:\dev\

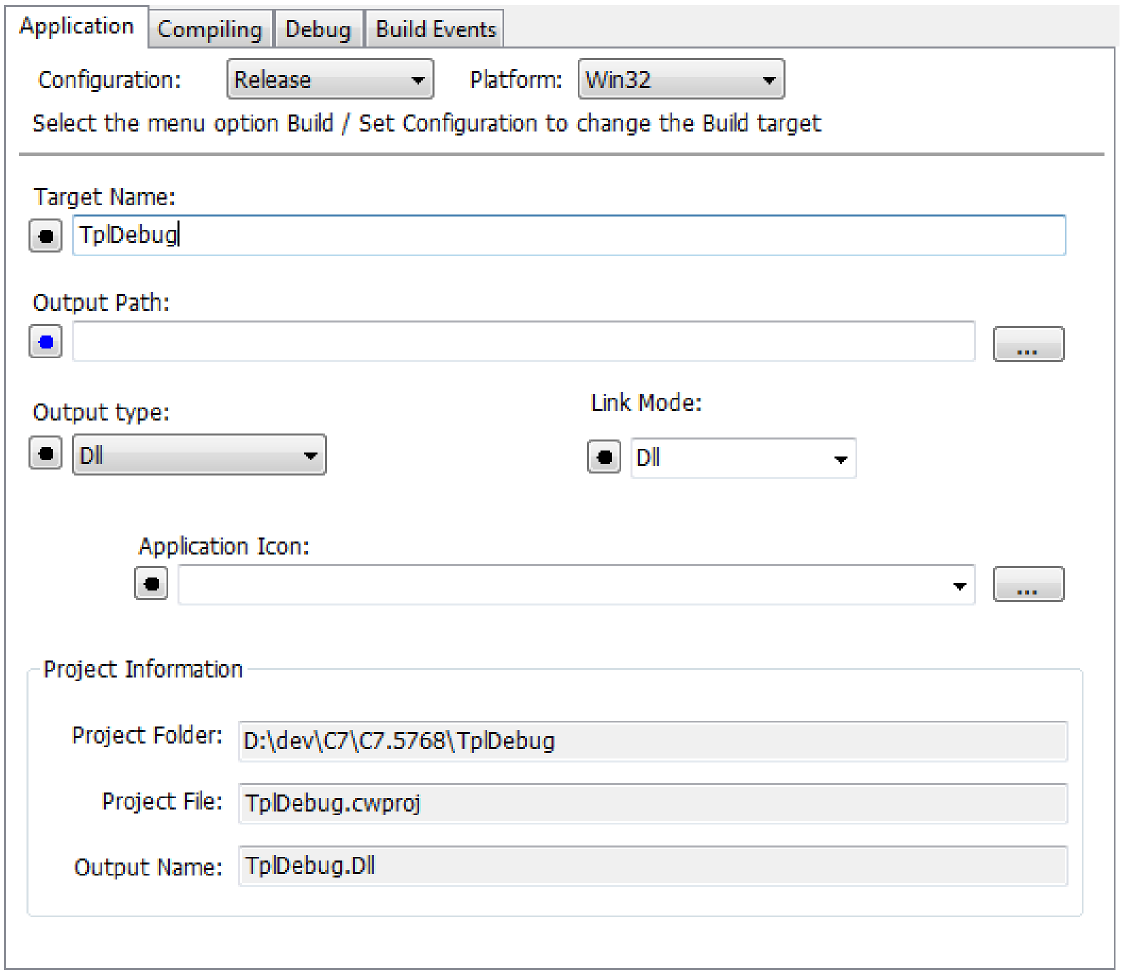
The program source remains the same as in the C6 version:

```
      PROGRAM
      MAP
        ODS(*CSTRING),Name('ODS')
        MODULE('Winapi')
          OutputDebugString(*CSTRING),PASCAL,RAW,NAME('OutputDebugStringA')
        END
      END

      CODE

    ODS          PROCEDURE(*CSTRING argMsg)
      CODE
      OutputDebugString(argMsg)
```

as does the EXP file:

```
    NAME 'TplDebug' GUI
     EXPORTS
       ODS      @?
     ; ODS@FRsc @?
```

Verify the project settings by right-clicking on the project (not the solution) in the Solution Explorer and choosing Properties. Figure 2 shows the Application tab.



Configuration: Release

Platform: Win32

Select the menu option Build / Set Configuration to change the Build target

Target Name: TplDebug

Output Path:

Output type: Dll

Link Mode: Dll

Application Icon:

Project Information

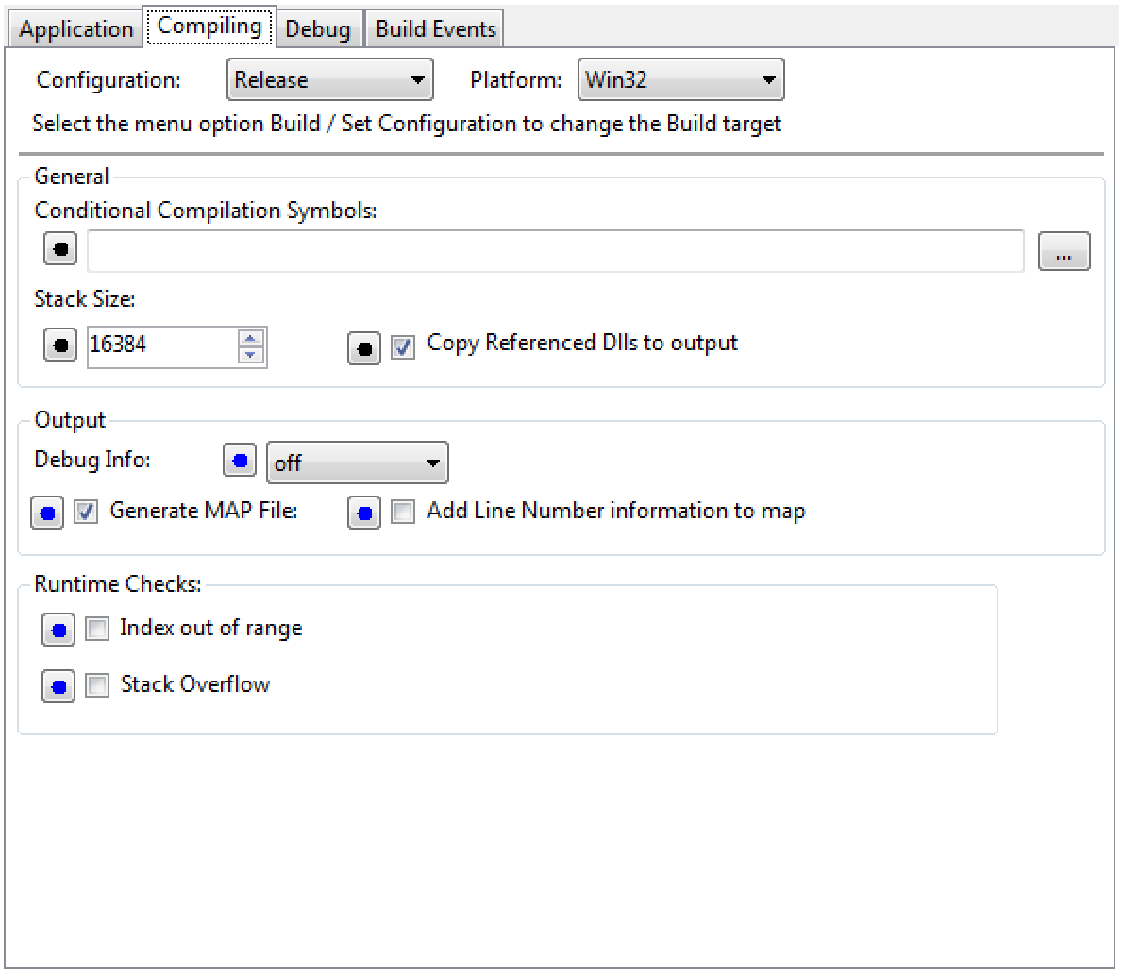
Project Folder: D:\dev\C7\C7.5768\TplDebug

Project File: TplDebug.cwproj

Output Name: TplDebug.Dll

Figure 2. The project settings, Application tab

And Figure 3 shows the Compiling tab.



Configuration: Release
Platform: Win32
Select the menu option Build / Set Configuration to change the Build target

General
Conditional Compilation Symbols:
Stack Size: 16384
☑ Copy Referenced Dlls to output

Output
Debug Info: off
☑ Generate MAP File:
☐ Add Line Number information to map

Runtime Checks:
☐ Index out of range
☐ Stack Overflow

Figure 3. Project settings, Compiling tab.

In both of these figures you can see the Release settings; you can choose it from the dropdown list, but to get it to stick you need to go to Build | Set Configuration and choose Release. But debug mode works just as well.

The template code

Again, the template code in Russ's article works fine, although I use a slightly different #Group wrapper:

```
#Group(%Trace,%Msg)
#RUNDLL('TplDebug.DLL','ODS',%Msg),WIN32
```

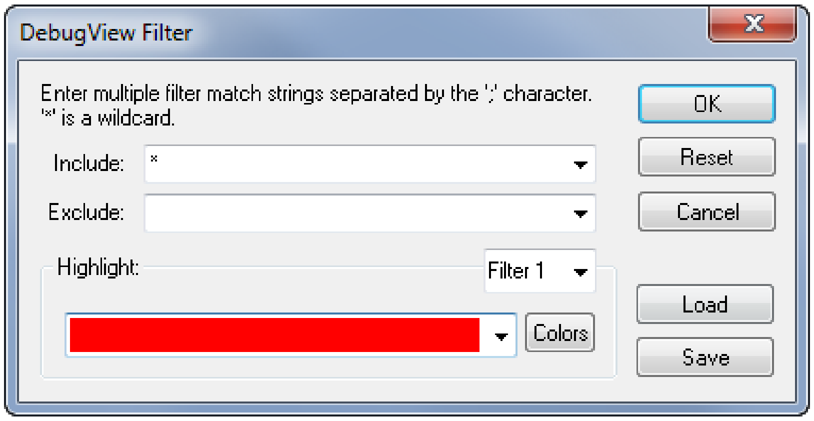
You can enhance your logging group in all sorts of ways. I often want to differentiate between routine tracing of logic and specific debug messages, so I have another group called %Debug:

```
#Group(%Debug,%Msg)
#RUNDLL('TplDebug.DLL','ODS','dbg ' & %Msg),WIN32
```

This group simply adds a 'dbg' string to the beginning of the statement so I can easily pick out the statements visually or with a filter in DebugView. You could implement all sorts of approaches, including logging levels.

Filtering and highlighting in DebugView

As Mark showed in his ClarionLive webinar, you can easily set highlighting rules in DebugView. But the window where you set those rules isn't all that intuitive, at least to me. From the DebugView menu choose Edit | Filter/Highlight, or press Ctrl-L.


DebugView Filter

Enter multiple filter match strings separated by the ';' character. '*' is a wildcard.

Include: *

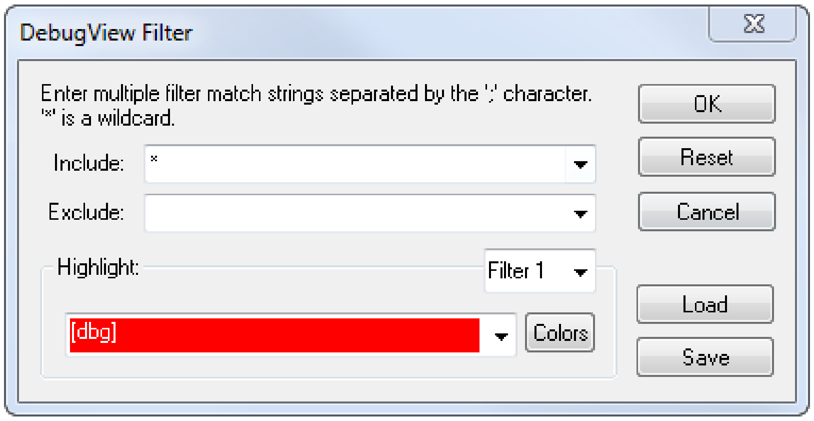
Exclude:

Highlight: Filter 1

Figure 4. Filtering and highlighting in DebugView

In the Include and Exclude fields you can specify one or more filter expressions. The * character is a wildcard, and you can have more than one expression in each field; just separate the expressions with a semicolon.

There are also 20 different highlight filters. Choose a filter from the drop-down list, and then (this is the critical bit) just type in the string on which you'd like to match. In Figure 5 I've entered the string [dbg] in the red highlight field to match my debug group's output.


DebugView Filter

Enter multiple filter match strings separated by the ';' character. '*' is a wildcard.

Include: *

Exclude:

Highlight: Filter 1

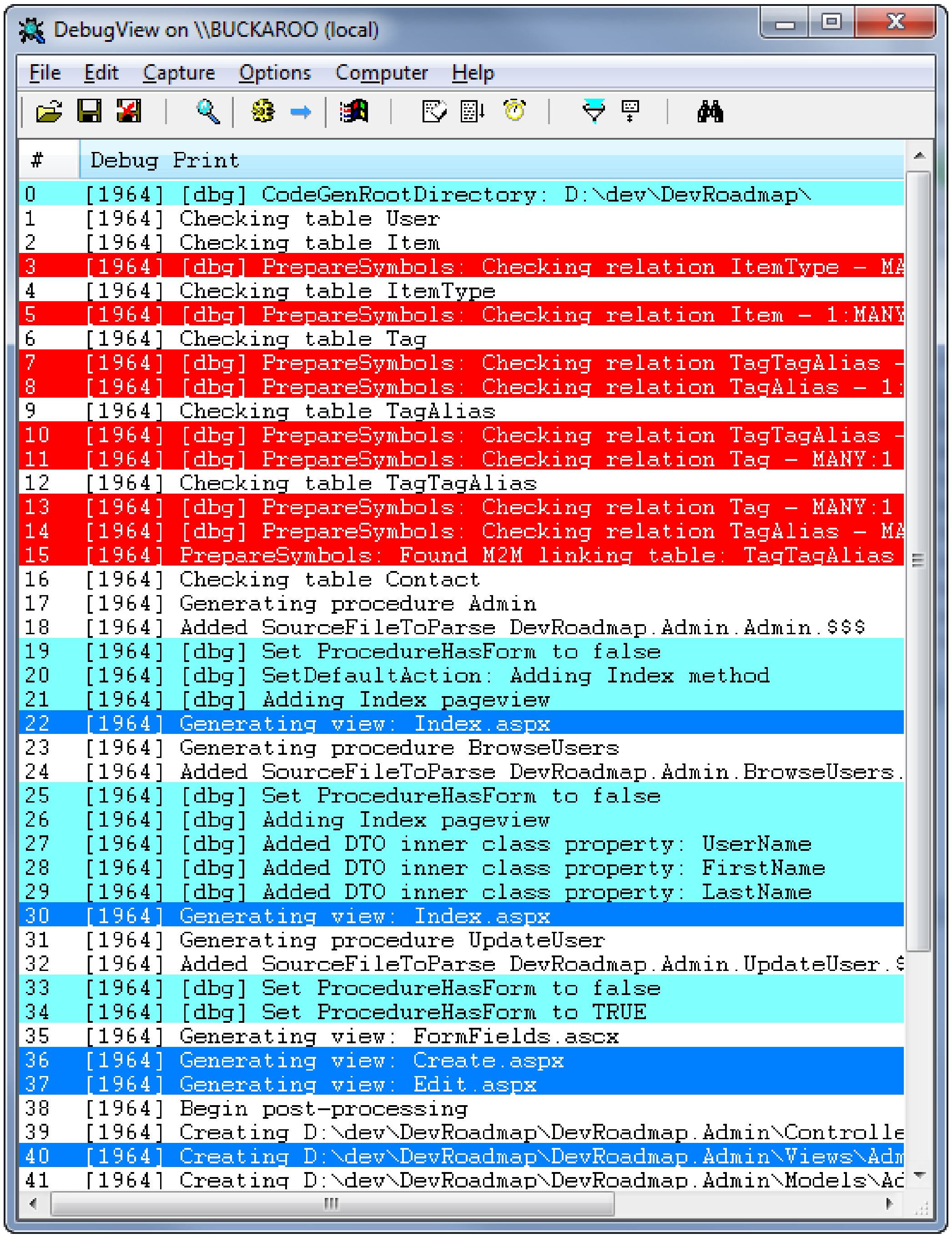
[dbg]

Figure 5. Highlighting [dbg]

You can customize foreground and background colors via the Colors button.

The highlighting is applied in filter order number. That is, if you have a line of text that matches more than one filter, the lower numbered filter will be applied. If you want a color to be applied as a default, but overridden if any more specific matches are found, put that rule at the end of the list.

In Figure 6 I've set a light blue rule matching [dbg], but this is the highest-numbered rule. The red color matches PrepareSymbols, so even though these lines are also marked with [dbg] they get the lower-numbered red coloring.


DebugView on \\BUCKAROO (local)
File Edit Capture Options Computer Help

```
#   Debug Print
0   [1964] [dbg] CodeGenRootDirectory: D:\dev\DevRoadmap\
1   [1964] Checking table User
2   [1964] Checking table Item
3   [1964] [dbg] PrepareSymbols: Checking relation ItemType - MA
4   [1964] Checking table ItemType
5   [1964] [dbg] PrepareSymbols: Checking relation Item - 1:MANY
6   [1964] Checking table Tag
7   [1964] [dbg] PrepareSymbols: Checking relation TagTagAlias -
8   [1964] [dbg] PrepareSymbols: Checking relation TagAlias - 1:
9   [1964] Checking table TagAlias
10  [1964] [dbg] PrepareSymbols: Checking relation TagTagAlias -
11  [1964] [dbg] PrepareSymbols: Checking relation Tag - MANY:1
12  [1964] Checking table TagTagAlias
13  [1964] [dbg] PrepareSymbols: Checking relation Tag - MANY:1
14  [1964] [dbg] PrepareSymbols: Checking relation TagAlias - MA
15  [1964] PrepareSymbols: Found M2M linking table: TagTagAlias
16  [1964] Checking table Contact
17  [1964] Generating procedure Admin
18  [1964] Added SourceFileToParse DevRoadmap.Admin.Admin.$$$
19  [1964] [dbg] Set ProcedureHasForm to false
20  [1964] [dbg] SetDefaultAction: Adding Index method
21  [1964] [dbg] Adding Index pageview
22  [1964] Generating view: Index.aspx
23  [1964] Generating procedure BrowseUsers
24  [1964] Added SourceFileToParse DevRoadmap.Admin.BrowseUsers.
25  [1964] [dbg] Set ProcedureHasForm to false
26  [1964] [dbg] Adding Index pageview
27  [1964] [dbg] Added DTO inner class property: UserName
28  [1964] [dbg] Added DTO inner class property: FirstName
29  [1964] [dbg] Added DTO inner class property: LastName
30  [1964] Generating view: Index.aspx
31  [1964] Generating procedure UpdateUser
32  [1964] Added SourceFileToParse DevRoadmap.Admin.UpdateUser.$
33  [1964] [dbg] Set ProcedureHasForm to false
34  [1964] [dbg] Set ProcedureHasForm to TRUE
35  [1964] Generating view: FormFields.ascx
36  [1964] Generating view: Create.aspx
37  [1964] Generating view: Edit.aspx
38  [1964] Begin post-processing
39  [1964] Creating D:\dev\DevRoadmap\DevRoadmap.Admin\Controlle
40  [1964] Creating D:\dev\DevRoadmap\DevRoadmap.Admin\Views\Adm
41  [1964] Creating D:\dev\DevRoadmap\DevRoadmap.Admin\Models\Ac
```

Figure 6. Coloring DebugView output

Source code and other resources

You can download the C7 project below. Other downloads you may want to get include:

- Mark Goldberg's current version of the Debuger [sic] class, which lets you output debug statements to DebugView from your Clarion code.
- DebugView, originally from SysInternals, now a Microsoft product.
- Russ Eggen's version with an ABC-compliant Debuger wrapper template

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

The Problem With Embeds, Part 1: What Went Wrong?

by Dave Harms

Published 2009-09-28

As Clarion developers we (well, most of us) live and die by embed code. Embeds are, after all, at the heart of Clarion's effectiveness. They're the primary way we add custom code to applications that are otherwise template-generated.

And if you're like most Clarion developers, you're probably using embed points the wrong way. Without realizing it you're making your applications more difficult to maintain, harder to debug, and almost impossible to document.

In this series of articles I'll analyze the embedded code in the Invoice example that ships with Clarion, and I'll show how significant portions of that application's embed code can be transformed into something that is maintainable, debuggable, even testable, and much easier to document.

The Clarion application architecture

We all use Clarion because in some way or another it makes our jobs as developers easier. There are still a few hand-coders in our midst who find sufficient benefit in the Clarion language itself, but for the rest of us it's all about the templates. And whether you're using ABC or Legacy (Clarion) templates, the great thing is that Clarion creates an *entire application framework* for you, largely based on the contents of your dictionary.

Application frameworks aren't talked about much in the Clarion world, mostly because we don't have to build them. They're supplied for us, and we've just come to accept them as the way things are.

The key features embodied in Clarion applications (leaving aside for the moment how they're built in the first place) include:

- FILE structures to model files/tables
- The database access grammar
- VIEWs for more sophisticated data access
- The browse/form approach to viewing and updating data
- Queue-based browse paging
- Reporting via the REPORT structure
- WINDOW structures with built in data binding
- The ACCEPT loop

The ABC library abstracts away some of this functionality, like the ACCEPT loop which is now hidden inside the WindowManager class. Similarly, the FileManager and RelationManager classes add a new layer on top of traditional data manipulation (although this abstraction is quite leaky, as when you have to issue a SET using the file access grammar before you can use the FileManager to retrieve data in key order).

The vast majority of Clarion applications in existence are AppGen creation; they embody these classic Clarion architectural features. Among other things that gives Clarion developers a common basis of discussion, and I don't think it's going too far to say that this shared application architecture is the main reason, or at least a very big reason, for the strength of the Clarion community over the last twenty years.

The problem with the Clarion architecture

But as useful as the Clarion architecture is, it has a problem. And that problem is going to grow in size as your applications grow in size and complexity.

The problem is embeds.

Not that there are embeds. We'd be lost without them. Embed points make it possible to plug our custom code into the almost any location within the standard Clarion architecture.

The problem is that embeds are almost always misused.

Example #1: The Invoice app

Here's an example from the shipping Clarion sample apps, specifically the Invoice app. The following is a listing of the embedded source from the UpdateDetail procedure, which lets the user add invoice line items:

```
PROCEDURE: UpdateDetail

 EMBED: %ProcedureRoutines 4000

 !Calculate taxes, discounts, and total cost
 !-----------------------------------------------------------------------
 CalcValues  ROUTINE
   IF DTL:TaxRate = 0 THEN
     IF DTL:DiscountRate = 0 THEN
       DTL:TotalCost = DTL:Price * DTL:QuantityOrdered
     ELSE
       LOC:RegTotalPrice = DTL:Price * DTL:QuantityOrdered
       DTL:Discount = LOC:RegTotalPrice * DTL:DiscountRate / 100
       DTL:TotalCost = LOC:RegTotalPrice - DTL:Discount
       DTL:Savings = LOC:RegTotalPrice - DTL:TotalCost
     END
   ELSE
     IF DTL:DiscountRate = 0 THEN
       LOC:RegTotalPrice = DTL:Price * DTL:QuantityOrdered
       DTL:TaxPaid = LOC:RegTotalPrice * DTL:TaxRate / 100
       DTL:TotalCost = LOC:RegTotalPrice + DTL:TaxPaid
     ELSE
       LOC:RegTotalPrice = DTL:Price * DTL:QuantityOrdered
       DTL:Discount = LOC:RegTotalPrice * DTL:DiscountRate / 100
```

```
        LOC:DiscTotalPrice = LOC:RegTotalPrice - DTL:Discount
        DTL:TaxPaid = LOC:DiscTotalPrice * DTL:TaxRate / 100
        DTL:TotalCost = LOC:DiscTotalPrice + DTL:TaxPaid
        DTL:Savings = LOC:RegTotalPrice - DTL:TotalCost
      END
    END

EMBED: %ProcedureRoutines 4000

  !Update InvHist and Products files
  !-----------------------------------------------------------------------
  UpdateOtherFiles ROUTINE

   PRO:ProductNumber = DTL:ProductNumber
   Access:Products.TryFetch(PRO:KeyProductNumber)
   CASE ThisWindow.Request
   OF InsertRecord
     IF DTL:BackOrdered = FALSE
       PRO:QuantityInStock -= DTL:QuantityOrdered
       IF Access:Products.Update() <> Level:Benign
         STOP(ERROR())
       END !end if
       INV:Date = TODAY()
       INV:ProductNumber = DTL:ProductNumber
       INV:TransType = 'Sale'
       INV:Quantity =- DTL:QuantityOrdered
       INV:Cost = PRO:Cost
       INV:Notes = 'New purchase'
       IF Access:InvHist.Insert() <> Level:Benign
         STOP(ERROR())
       END !end if
     END !end if
   OF ChangeRecord
     IF SAV:BackOrder = FALSE
       PRO:QuantityInStock += SAV:Quantity
       PRO:QuantityInStock -= NEW:Quantity
       IF Access:Products.Update() <> Level:Benign
         STOP(ERROR())
       END
       InvHist.Date = TODAY()
       INV:ProductNumber = DTL:ProductNumber
       INV:TransType = 'Adj.'
       INV:Quantity = (SAV:Quantity - NEW:Quantity)
       INV:Notes = 'Change in quantity purchased'
       IF Access:InvHist.Insert() <> Level:Benign
         STOP(ERROR())
       END !end if
     ELSIF SAV:BackOrder = TRUE AND DTL:BackOrdered = FALSE
       PRO:QuantityInStock -= DTL:QuantityOrdered
       IF Access:Products.Update() <> Level:Benign
         STOP(ERROR())
       END !end if
       INV:Date = TODAY()
       INV:ProductNumber = DTL:ProductNumber
```

```
      INV:TransType = 'Sale'
      INV:Quantity =- DTL:QuantityOrdered
      INV:Cost = PRO:Cost
      INV:Notes = 'New purchase'
      IF Access:InvHist.Insert() <> Level:Benign
        STOP(ERROR())
      END !end if
    END ! end if elsif
  OF DeleteRecord
    IF SAV:BackOrder = FALSE
      PRO:QuantityInStock += DTL:QuantityOrdered
      IF Access:Products.Update() <> Level:Benign
        STOP(ERROR())
      END
      INV:Date = TODAY()
      INV:ProductNumber = DTL:ProductNumber
      INV:TransType = 'Adj.'
      INV:Quantity =+ DTL:QuantityOrdered
      INV:Notes = 'Cancelled Order'
      IF Access:InvHist.Insert() <> Level:Benign
        STOP(ERROR())
      END  !End if
    END !End if
  END !End case
```

EMBED: %WindowManagerMethodCodeSection Init (),BYTE 6500

```
  !Initializing a variable
  SAV:Quantity = DTL:QuantityOrdered
  SAV:BackOrder = DTL:BackOrdered
  CheckFlag = False

EMBED: %WindowManagerMethodCodeSection Init (),BYTE 8030

  IF ThisWindow.Request = ChangeRecord OR ThisWindow.Request = DeleteRecord
    PRO:ProductNumber = DTL:ProductNumber
    Access:Products.TryFetch(PRO:KeyProductNumber)
    ProductDescription = PRO:Description
  END

EMBED: %WindowManagerMethodCodeSection Init 4000

   !Updating other files
    DO UpdateOtherFiles

EMBED: %WindowManagerMethodCodeSection 8500

  ! After lookup and out of stock message
  IF GlobalResponse = RequestCompleted
    DTL:ProductNumber = PRO:ProductNumber
    ProductDescription = PRO:Description
    DTL:Price = PRO:Price
    LOC:QuantityAvailable = PRO:QuantityInStock
    DISPLAY
```

```
IF LOC:QuantityAvailable <= 0
  CASE MESSAGE('Yes for BACKORDER or No for CANCEL',|
               'OUT OF STOCK: Select Order Options',ICON:Question,|
                 BUTTON:Yes+BUTTON:No,BUTTON:Yes,1)
  OF BUTTON:Yes
    DTL:BackOrdered = TRUE
    DISPLAY
    SELECT(?DTL:QuantityOrdered)
  OF BUTTON:No
    IF ThisWindow.Request = InsertRecord
      ThisWindow.Response = RequestCancelled
      Access:Detail.CancelAutoInc
      POST(EVENT:CloseWindow)
    END !If
  END !end case
END !end if
IF ThisWindow.Request = ChangeRecord
  IF DTL:QuantityOrdered < LOC:QuantityAvailable
    DTL:BackOrdered = FALSE
    DISPLAY
  ELSE
    DTL:BackOrdered = TRUE
    DISPLAY
  END !end if
END !end if
IF ProductDescription = ''
  CLEAR(DTL:Price)
```

```
      SELECT(?CallLookup)
    END
    SELECT(?DTL:QuantityOrdered)
  END

EMBED: %ControlEventHandling ?DTL:QuantityOrdered Accepted 8800

  !Initializing a variable
  NEW:Quantity = DTL:QuantityOrdered
   !Low stock message
  IF CheckFlag = FALSE
    IF LOC:QuantityAvailable > 0
      IF DTL:QuantityOrdered > LOC:QuantityAvailable
        CASE MESSAGE('Yes for BACKORDER or No for CANCEL',|
                     'LOW STOCK: Select Order Options',ICON:Question,|
                      BUTTON:Yes+BUTTON:No,BUTTON:Yes,1)
        OF BUTTON:Yes
          DTL:BackOrdered = TRUE
          DISPLAY
        OF BUTTON:No
          IF ThisWindow.Request = InsertRecord
            ThisWindow.Response = RequestCancelled
            Access:Detail.CancelAutoInc
            POST(EVENT:CloseWindow)
          END !
        END !end case
      ELSE
        DTL:BackOrdered = FALSE
        DISPLAY
      END !end if Detail
    END !End if LOC:
    IF ThisWindow.Request = ChangeRecord
      IF DTL:QuantityOrdered <= LOC:QuantityAvailable
        DTL:BackOrdered = FALSE
        DISPLAY
      ELSE
        DTL:BackOrdered = TRUE
        DISPLAY
      END !end if
    END !end if
    CheckFlag = TRUE
  END !end if

EMBED: %ControlEventHandling ?DTL:QuantityOrdered 6000

  !Calculate all totals
  DO CalcValues
```

To be fair, this is actually a pretty old example, and I don't think it's one SoftVelocity has ever put forward as an best practices standard. It does, however, look a lot like a whole lot of Clarion code I've seen (and written), and it's probably pretty familiar to you as well.

Some of that code is appropriate for embeds. After all, where else are you going to respond to the user clicking a button, or move some data to the screen?

So, what's wrong with it? A few things:

- It's difficult to code and modify. You can't edit this code all by itself, the above listing notwithstanding; instead, you're dealing with different embed points at different locations in the generated source
- It isn't reusable. You can't easily take this code and use it somewhere else, say in a modified version of this form as required by a different client.
- It isn't testable. You really have no way of knowing if, say, the tax calculation code works other than to try out the form.
- It's difficult to understand.

There's something quite valuable buried in all this code. It's not the database access; it's not the way the screen is updated. The valuable bit is *the logic behind a particular way of handling invoicing*.

Business logic: the real value of your application

Think of your application as made up of three parts: the user interface, the data store (TPS files, SQL tables) and the business logic. That's a slight oversimplification, to be sure, because there can be some aspects of your application that don't fit nearly into those categories. But it's a useful categorization because it helps to focus attention on the part of your application that is truly unique.

What makes your application valuable to your clients? Is it the user interface (UI)? Possibly, but unless you write your own UI layer then chances are other applications out there also use similar controls, if in different arrangements. For most of us (especially as Clarion developers), the UI is not what makes our customers open their wallets.

Is it the data layer? If you use TPS files you have certain advantages and disadvantages over developers using other database systems, but unless you're selling specialized data access or data mining where the key is the speed and power of the database, it's not going to be your data layer that gives your application its unique value.

In almost all cases what gives a business application unique value is the *business logic*. That logic has to fit with how your customers do their business; the more flexible, the more reliable your business logic is, the more robust your application. Imagine the Invoice.app is your own product (stay with me here). When you create invoices, applying taxes and discounts as appropriate to your industry, that's your business logic. When you manage inventory movement according to the unique requirements of your shipping system, that's your business logic.

And here's the thing: for the most part, Clarion isn't much help when it comes to writing code that embodies all that uniquely valuable business logic.

That's right. Clarion is *next to useless when it comes to creating the most valuable part of your application*. It's no better than any other environment, and actually worse than some!

But Clarion is enormously useful at handling all the grunt work of creating menus, browses, forms, reports etc. It does all that work *so you can devote most of your energy to writing the stuff that really matters*. That's Clarion's true competitive advantage.

So, back to the Invoice app for a moment.

Somewhere, buried in that embed code for the UpdateDetail procedure, is a business process for handling invoicing. A quick glance wouldn't tell you that, and a closer look would probably leave you a little confused (at least it does me).

Extracting business logic

My task, in this series of articles, is to explore the ways that business logic can be teased out of the embed point and put into some code that's easy to understand, easy to maintain and modify, and easy to test.

Of course, I'll still need those embeds so I can call that business logic, wherever it ends up. But the point of this exercise is to figure out how business logic can be coded for maximum benefit. Stuffing it all in embed points just has too many drawbacks.

I won't, however, be tackling the Invoice example until a little later on in this series. I've showed it to you because I think it's a fairly typical example, and I want you to know that's where these articles are headed. But it's also problematic because it's so tightly tied to the database.

Instead, I'm going to ease into this with a simpler example. And it's kind of a funny one.

Example #2: Embedded code to find embedded code

As I began to plan out this series of articles I realized very early on that I'd need a way to get a report on the embed code used by all the procedures in any given application. It may be that there's a template solution to this problem; I'm not aware of any such template. Rather, the approach I've used for extracting embed information is to export a TXA and then parse that TXA, looking for the embeds.

This isn't an easy task. But as it happens I took it on two and a half years ago when I ran a series of articles on the most popular embed points, based on data extracted from a number of reader-submitted TXAs. That code wasn't specifically concerned with the contents of the embeds, just whether or not they were used. But it wasn't that hard to adapt the code to extract the embedded source itself.

And here's the funny part. I wrote that original TXA parser as embedded code. Ha ha.

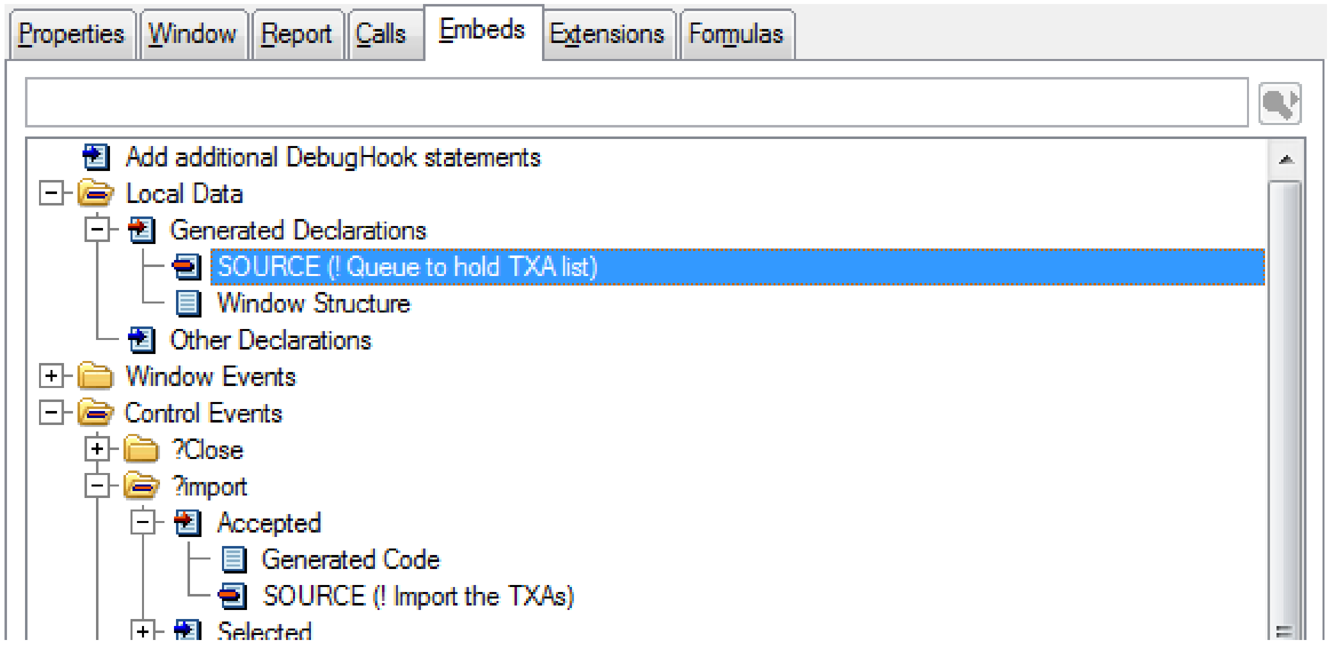
It's a mess.

Oh, the code isn't so bad. I only found a couple of bugs in it when I reworked it for *this* article series. But like the Invoice app, my embedded version of the TXA parser has some serious problems, including an awkward location, an inability to test, and tight binding to a particular database, making reuse in other situations impossible without significant modifications. Of course, at the time I couldn't imagine another place where I'd want to reuse this code, so why did I care?

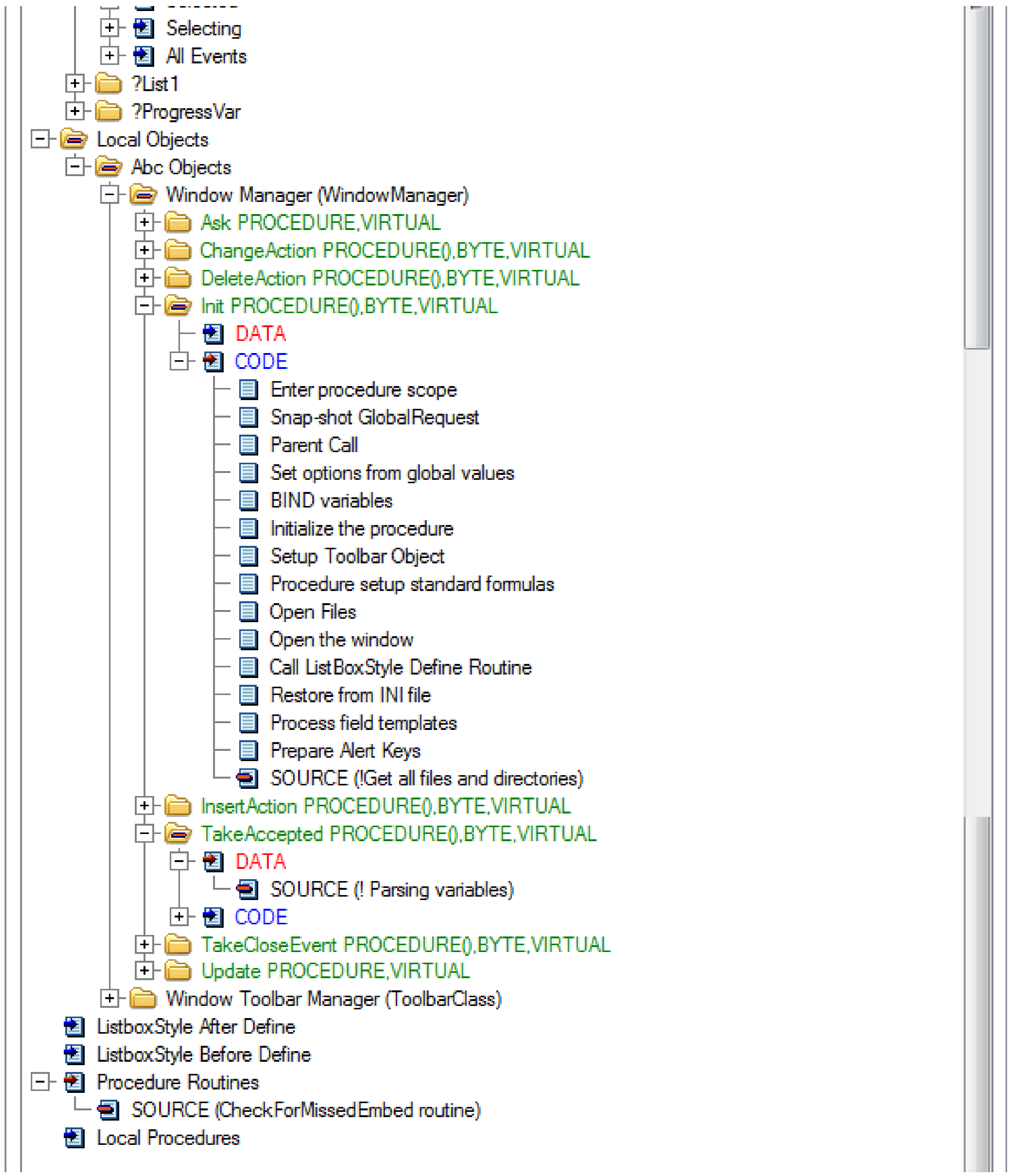
Well, now I care.

Just remember - this is probably happening to you too, right now. You've got all sorts of code tucked away in your embed points, and much of it *really doesn't belong there*.

Figure 1 shows a screen shot of the embed list (slightly abbreviated) for the TXA parser procedure.



Add additional DebugHook statements
Local Data
Generated Declarations
SOURCE (! Queue to hold TXA list)
Window Structure
Other Declarations
Window Events
Control Events
?Close
?import
Accepted
Generated Code
SOURCE (! Import the TXAs)
Selected



```
        Selecting
        All Events
    ?List1
    ?ProgressVar
Local Objects
    Abc Objects
        Window Manager (WindowManager)
            Ask PROCEDURE,VIRTUAL
            ChangeAction PROCEDURE(),BYTE,VIRTUAL
            DeleteAction PROCEDURE(),BYTE,VIRTUAL
            Init PROCEDURE(),BYTE,VIRTUAL
                DATA
                CODE
                    Enter procedure scope
                    Snap-shot GlobalRequest
                    Parent Call
                    Set options from global values
                    BIND variables
                    Initialize the procedure
                    Setup Toolbar Object
                    Procedure setup standard formulas
                    Open Files
                    Open the window
                    Call ListBoxStyle Define Routine
                    Restore from INI file
                    Process field templates
                    Prepare Alert Keys
                    SOURCE (!Get all files and directories)
            InsertAction PROCEDURE(),BYTE,VIRTUAL
            TakeAccepted PROCEDURE(),BYTE,VIRTUAL
                DATA
                    SOURCE (! Parsing variables)
                CODE
            TakeCloseEvent PROCEDURE(),BYTE,VIRTUAL
            Update PROCEDURE,VIRTUAL
        Window Toolbar Manager (ToolbarClass)
    ListboxStyle After Define
    ListboxStyle Before Define
Procedure Routines
    SOURCE (CheckForMissedEmbed routine)
Local Procedures
```

Figure 1. The original TXAParser procedure

There are just five embeds in the parser procedure. First, there's some global data:

```
! Queue to hold TXA list
txaq          QUEUE(File:queue),pre()
              END
state         LONG(0)
LineNo        long
```

Then there's a line of code in the Init method to load up the queue of TXAs to import:

```
  !Get all files and directories
  DIRECTORY(txaq,'*.TXA',ff_:NORMAL)
```

There's one small routine that gets called regularly by the main import code, and which is really just a bit of debugging code:

```
CheckForMissedEmbed routine
  if sub(txt:rec,1,8) = '[SOURCE]'
    db.out('*** MISSED EMBED at line ' & lineno |
      & ', state ' &  state & ' ***')
  end
```

There's an embed for variables used in the parsing code:

```
x                    long
vars                 group,pre()
procname                  string(200)
```

```
procFromABC              string(60)
procCategory             string(60)
embedname                string(60)
embedPriority            long
embedParam1              string(200)
embedParam2              string(200)
embedParam3              string(200)
whenLevel                byte
                   end
LastProcName             like(procname)
lastEmbedName            string(500)
currembedname            string(500)
```

And finally there's one very large block of code to import the TXAs, which I won't reproduce in its entirety. It's basically a state engine that process the TXA in a loop and figures out where the embeds are and what they contain:

```
access:TextFile.Open()
Access:TextFile.UseFile()
set(TextFile)
state = 0
lineNo = 0
clear(procname)
clear(lastprocname)
clear(lastembedname)
clear(currembedname)
LOOP
   next(TextFile)
   if errorcode() then break.
```

```
lineNo += 1
CASE state
OF 0 ! search for the start of a module or procedure, or an embed
  if sub(txt:rec,1,11) = '[PROCEDURE]'
    clear(vars)
    state = 10
  elsif sub(txt:rec,1,8) = '[MODULE]'
    clear(vars)
    procName = '[MODULE]'
  elsif sub(txt:rec,1,7) = 'EMBED %'
    embedName = sub(txt:rec,7,len(txt:rec))
    state = 30
  elsif sub(txt:rec,1,8) = '[SOURCE]'
    state = 50

  end
OF 10 ! get procedure name details

! OF cases for a bunch of other states



of 50  ! Add the procedure and embed records
  if sub(txt:rec,1,8) = 'PRIORITY'
    embedPriority = sub(txt:rec,10,len(txt:rec))
    if lastprocname <> procname
      clear(EMP:record)
      EMP:Proc = procname
```

```
                EMP:ProcFromABC = ProcFromABC
                EMP:ProcCategory = ProcCategory
                EMP:EmbedAppID = EMA:EmbedAppID
                Access:EmbedProc.Insert()
                lastprocname = procname
            end
            ! Add the embed record
            currEmbedName = clip(embedName) & clip(embedparam1) & |
                clip(embedparam2) & clip(embedparam3) & embedpriority
            if currEmbedName <> lastEmbedName
                ! Add the embed record yada yada
            end
            state = 51
        end

    ! OF cases for a bunch of other states

    END
end
?progressVar{prop:progress} = records(txaq)
setcursor()
Access:TextFile.Close()
```

There's a ton more code, but you get the idea. This code is completely dependent on a specific database structure, and it references a control on the current window. It's sprinkled throughout a bunch of generated code. Those are all problems, to some degree. But mostly it's just hard to reuse this code somewhere else.

Making the TXA parser reusable

Now, how would I go about taking that parsing code and making it into something that could be used in that original embed analysis app as well as in the new embed extraction tool, the one that creates the listing from the Invoice app?

I really had three choices:

1. Put the code in a procedure
2. Put the code in a class
3. Put the code in a template

You've probably heard the saying "the second time, it's a template." The idea is that if you find yourself writing the same code more than once, it really ought to go into a template.

The second time, it's not a template

The template advocacy statement I like is this one: "As long as your code isn't critical, unique business logic, the second time it goes in a template. Otherwise it goes into its own source file."

Templates aren't an optimal solution for most business logic, or really for any complex logic. If you try to treat the template language as a business programming language, you'll quickly become frustrated by the quirky syntax, the need to mix Clarion language statements with template language statements, the difficulty of debugging, etc.

The template language's job isn't to take the place of the Clarion language, it's to make it easy to generate repetitive Clarion code. And core business logic is generally not repetitive.

Clearly I'm going to write my core business logic as Clarion code. So that leaves procedures and classes.

Which approach did I choose? Tune in next time to find out....

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

Reader Comments

Posted on Monday, September 28, 2009 by Gregory Bailey

Hi Dave,

I find the thoughts behind this article interesting.

I believe I will use it and the follow up articles as the basis of discussion at my next user group meeting.

Posted on Monday, September 28, 2009 by Dave Harms

Cool - let me know how it goes, Greg.

Dave

Add a comment

Clarion Magazine

Making TPS Superfiles

by Steven Parker

Published 2009-09-28

I think I've demonstrated, in previous articles (Topspeed Superfiles and Superfiles and NAME), that Topspeed "superfiles" are nothing but regular Topspeed files. Or, more accurately, I've demonstrated that "regular" TPS files are superfiles but without multiple tables. Structurally, TPS files are no different than MS SQL files: one physical file containing multiple logical files (called "tables").

I showed how superfiles are amenable to all the tricks I might conceivably want to play with their Name attribute. The name of the physical file on disk and the names of the internal tables can each be held in a variable. And, as is always true when using variable file names, all that is required is correctly priming the variables before the first attempt to open the file.

I also discussed how superfiles got, at least initially, a less than stellar reputation. Terry Mullican observed in response to the first article:

> We used TPS Superfiles for more than ten years on two vertical market products with great success. I never understood all the people who did not use them. It keeps all your files together and makes for a nice clean install and maintenance of your files.

Another stalwart Clarioneer apparently was convinced of the utility of superfiles and asked how to created them from existing TPS files:

> Suppose I have an existing database with a whole bunch of TPS file and I want to convert them to be inside a super file?
>
> Is there a magic bullet or do I make a SuperDestination file and then write a series of one-table conversions to boot strap the SuperDestination into existence?

Creating new superfiles from existing TPS files is my subject this evening.

The hard way

We're programmers. We write programs. So, it comes as no surprise that the first thing a Clarioneer thinks about is writing a conversion program.

One way is to let Clarion create a conversion program for me, an option that has been available since Clarion 3. In 6.3 or earlier, and with the dictionary open, click File | Create Conversion Program. There is an option to do a single table or multiple tables conversion in Clarion 6.3. In Clarion 7, from the dictionary editor, right click on the file in the Dictionary Explorer and there are options for creating a conversion program (the multiple tables option does not appear here - it may be elsewhere but I haven't gone looking for it).

I have used the Create Conversion Program in CDD and C5.5, thus before the multi-table option was available. The key to making this work is that there have to be two dictionary entries, one for each version of the file. Typically, at least in the past, this meant two dictionaries (even when not actually required by the IDE).

The conversion program is a small source file. Set as the current Project, the CLW (CLA in DOS) will compile to a small EXE. The source file contains both file layouts, each declaration using the Name attribute. The Code section assigns the Names, loops through the file designated as "source" and assigns records to the "target" file.

For example, to convert a Btrieve file (from the DOS version of the program) to a Topspeed file (in the Windows version), the source file might look something like:

```
Data
ABADDONF        FILE,PRE(INP),DRIVER('Btrieve')|
                   ,CREATE,NAME(NameAtt1) ! Add on Field
AddOnFieldKey     KEY(+INP:AddOnField),NOCASE
RECORD          RECORD
AddOnField        STRING(3)
AOFDesc           STRING(20)
AOFAmount         DECIMAL(8,2)
```

```
                  . .

ABADDONF1         FILE,PRE(OUT),DRIVER('TOPSPEED')|
                     ,CREATE,NAME(NameAtt2) ! Add on Field
AddOnFieldKey       KEY(+OUT:AddOnField),NOCASE
RECORD              RECORD
AddOnField            Short
AOFDesc               STRING(20)
AOFAmount               DECIMAL(8,2)
                  . .

CODE

  NameAtt2 = 'abaddonf.tps'
  CREATE(ABADDONF1)
  NameAtt1 = 'ABADDONF.DAT'
  If ~Exists(NameAtt1) then Return.

  If UPPER(Nameatt1) = UPPER(Nameatt2)
    Path = Nameatt2
    Rtn = FnSplit( Path, drive, dir, name, ext )
    Nameatt2 = drive & dir & 'NEWDATA\' & name & ext
    Directory = drive & dir & 'NEWDATA'
    Rtn = MKDIR(DIRECTORY)
  END

  OPEN(ABADDONF)
  DO CheckError
  OPEN(ABADDONF1)
  DO CheckError
  stream(abaddonf)
  stream(abaddonf1)
  Records# = 0
  Convert# = Records(ABADDONF)
  SET(ABADDONF)
  LOOP
    NEXT(ABADDONF)
    IF ERRORCODE() = 33 THEN BREAK.
    DO CheckError
    DO LoadRecord
    APPEND(ABADDONF1)
    DO CheckError
    Records# += 1
    SHOW(12,20,'Addons Converted: '&FORMAT(Records#,@n9))
  END
  SHOW(13,20,'Now Building Key Files')
  flush(abaddonf)
  flush(abaddonf1)
  BUILD(ABADDONF1)
  Close(abaddonf)
  Close(abaddonf1)


LoadRecord ROUTINE
 Clear(OUT:Record)


  OUT:AddOnField          = INP:AddOnField
```

```
        OUT:AOFDesc          = INP:AOFDesc
        OUT:AOFAmount          = INP:AOFAmount
```

Not rocket science. But neither flexible nor easily adapted to multiple files in a single go.

Conversion program for multiple tables

The multi-table conversion option is available only in Clarion 6. Either I never noticed it or have forgotten about it. And it certainly looks very promising for the task currently at hand.

To test it, I took Superfiles.DCT from the previous articles and upgraded it to C6 (originally, it was 5.5). I then saved it to another name so that I would have a source and a target dictionary.

In this second dictionary, I removed the Full Pathname property which tells Clarion that I am using a superfile. So, I have a standard dictionary and my original dictionary with the same files designated as superfiles.

I opened the standard dictionary and selected the multi-table conversion menu option. This invoked the synchronizer.

I have not used the synchronizer much and the few times I did, I had inconsistent success. The attempt to create a single table to superfile conversion program was no exception. The file Names were incorrect and the Source and Target columns appeared to be backward from what I wanted. Further, no PRJ or CLW was generated when I clicked "OK."

Therefore, I have to leave further exploration of this option to someone more familiar with the synchronizer.

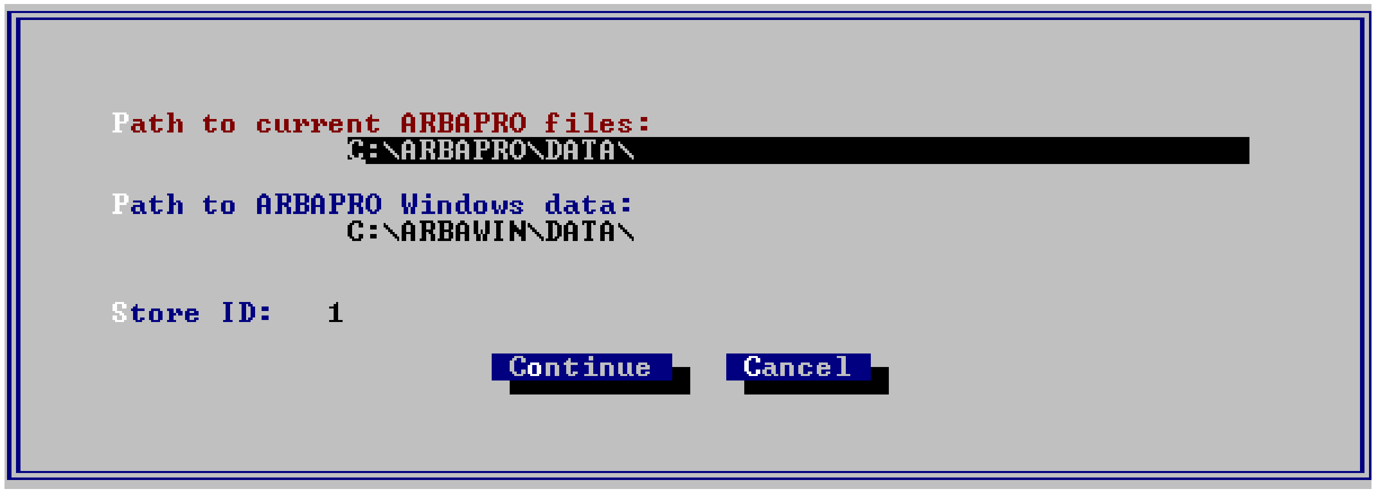
A less/more hard way

When I first started using the single file conversion programs Clarion generated for me, I found that they were limited. I could not specify source and target directories.

Also, I needed to convert an entire set of files, multiple files. At that time, should I have been able to get it to work for me, the multiple conversion options was not available.

So, I created a program that contained one procedure for each file I wanted to convert. I "imported" the code generated for me. But, I added some variables for the source and target paths.

When the user starts my program, they are required to tell me where the DOS (source) files are and where the Windows (target) files are to go.



```
Path to current ARBAPRO files:
             C:\ARBAPRO\DATA\

Path to ARBAPRO Windows data:
             C:\ARBAWIN\DATA\

Store ID:   1
```

Figure 1. User interface for paths

With this information, a slight edit to the generated code ensures that I am reading the correct file and writing to the correct place:

```
NameAtt2 = Clip(winPath) & 'abaddonf.tps'
CREATE(ABADDONF1)
NameAtt1 = Clip(dosPath) & 'ABADDONF.DAT'
```

(the post-processing code for the screen shown in Figure 1 ensures that the final character in each of the path variables is a back slash; it also checks that the expected files exist in the source path).

There are two main variations on this method.

1. If I use the "old" dictionary as the starting point, I can create a Process procedure for each file I want to convert. Because the "old" dictionary is the starting point, I do not have to cut and paste the layout for the source file. I do have copy, paste and modify the target file layout. The user gets a progress bar.
2. If I use the "new" dictionary as the starting point, I cannot use a Process (the Process template assumes that its Primary File

is in the DCT). So, here, I do have to copy and paste both file layouts.

Using a Window procedure, I can have a progress bar. But I have to manage it myself (see A Progress Bar For Multiple Processes). Alternately, I can simply have a record counter on the window.

Either way, I'll get the job done. But it is a lot of work. One typo and things go seriously awry. For one-off conversions, the default conversion programs are not bad. For a set of files, either of these methods really are a lot of work. For merging a set of files into a superfile, while writing this kind of program will work, there just has to be something better.

Something better

There is something better than hacking at, cloning or duplicating generated conversion source, and it has been part of the Clarion environment for years. It is the Topcopy utility. Topcopy, however, does not yet appear to be available as part of the C7 distro.

Topcopy will create the (new) superfile and allows me to add, import individual files into its tables.

Open Topcopy and create the new physical file in the left frame.

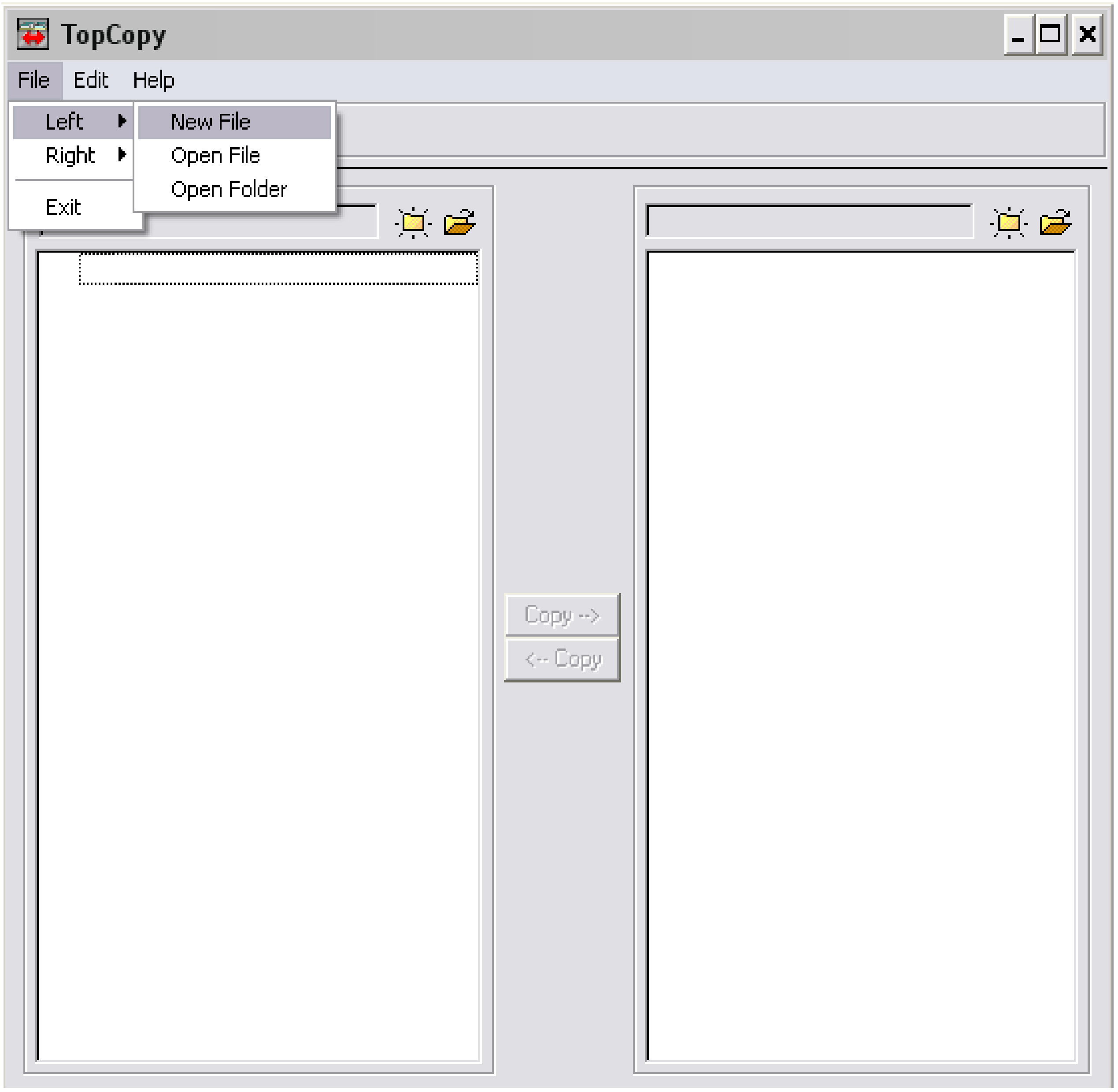


Figure 2. Prepare for physical file creation

Navigate to the target directory and specify the DOS file name.

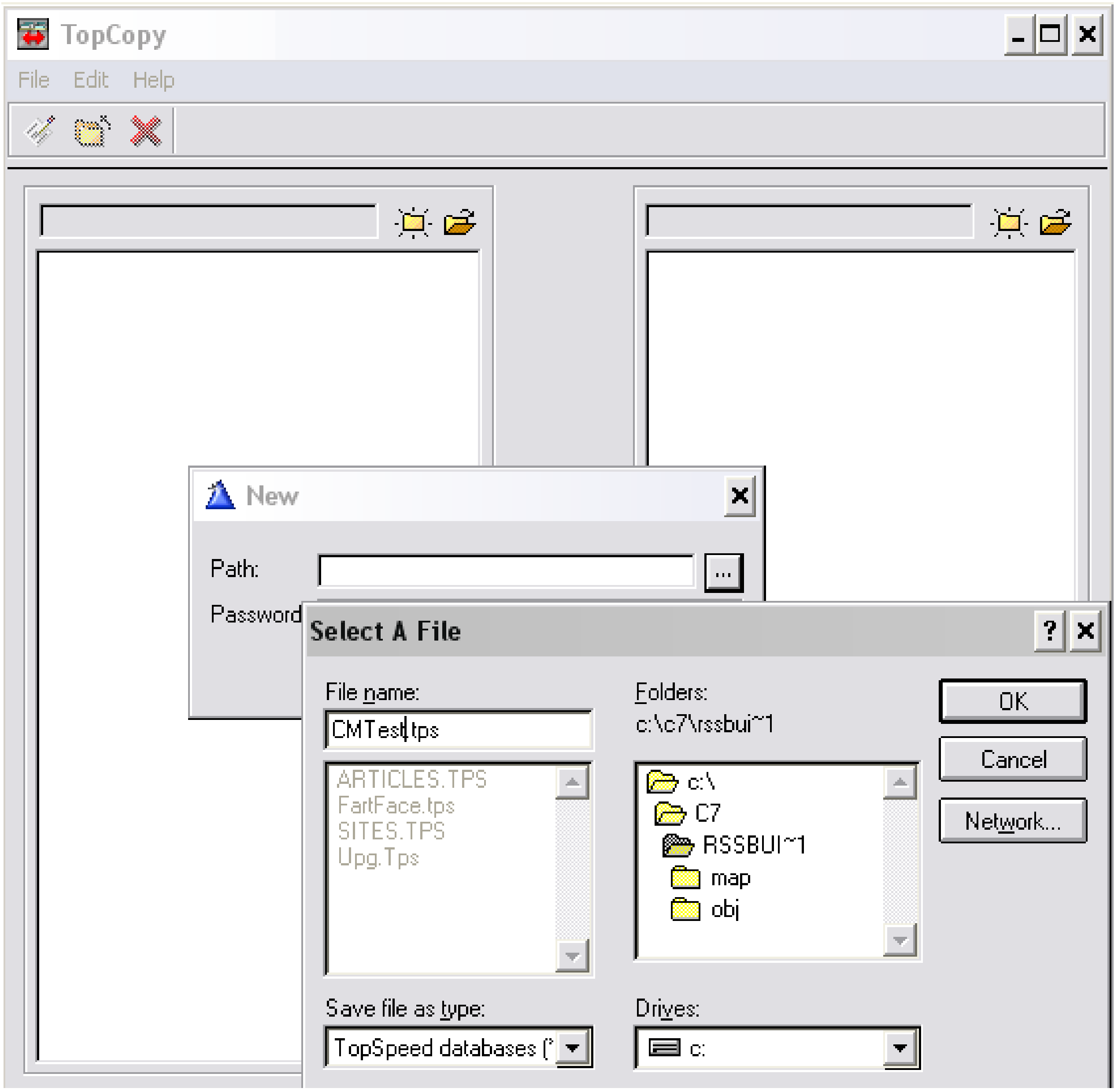
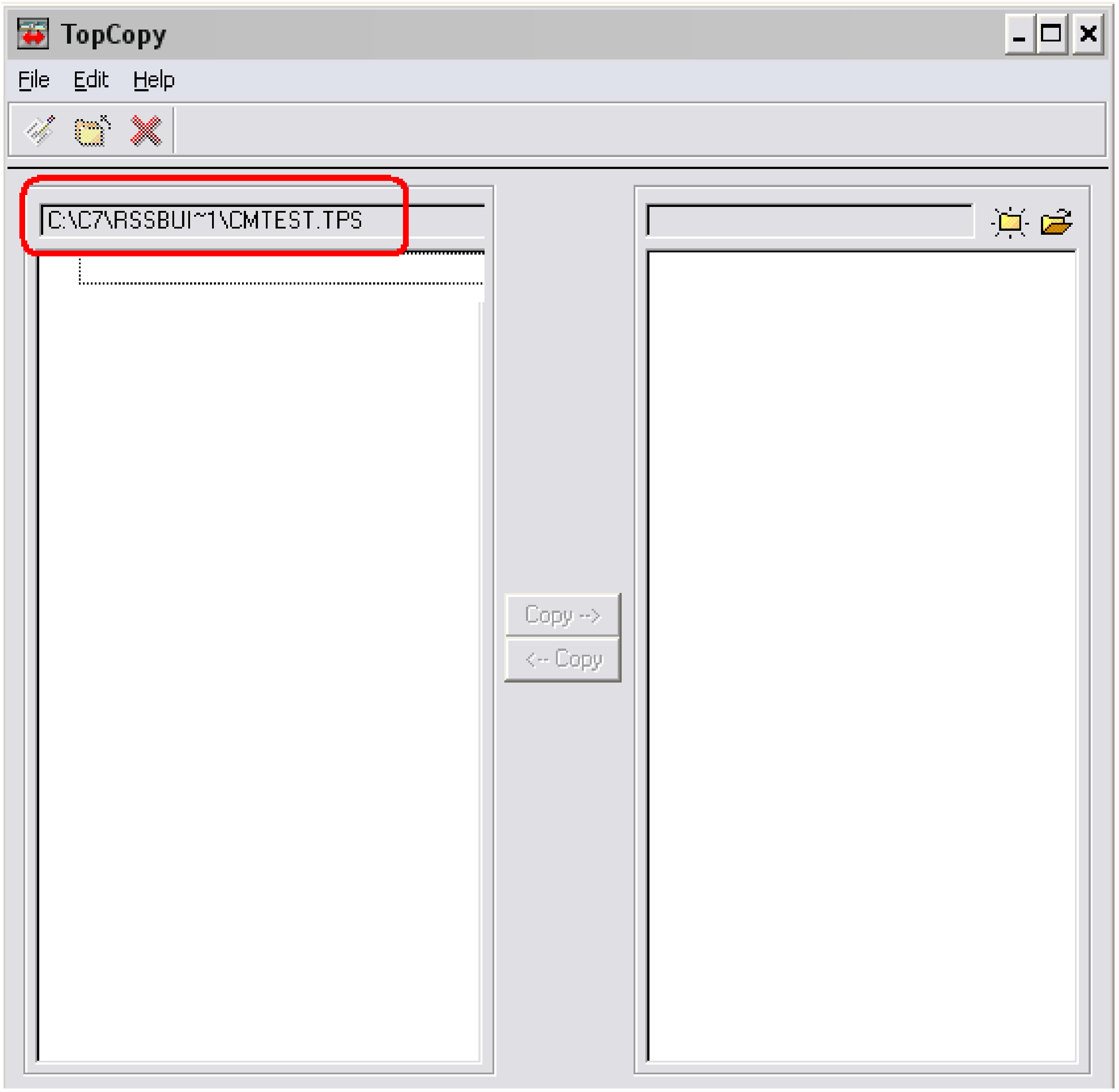


Figure 3. Specifying the target file

(N.B.: some of the screen shots are from Topcopy for Clarion 5.5, some are from Clarion 6 Topcopy. The screen shots that know about long file names are from 6. Use the C6 version.)

And, Topcopy knows about the new file.



C:\C7\RSSBUI~1\CMTEST.TPS

Figure 4. Topcopy "sees" the new file

Windows Explorer will also show the container file at this point.

Now I need to specify the first file to import. I will put this in the right frame. Because it is an existing file, I want to Open it instead of creating it.

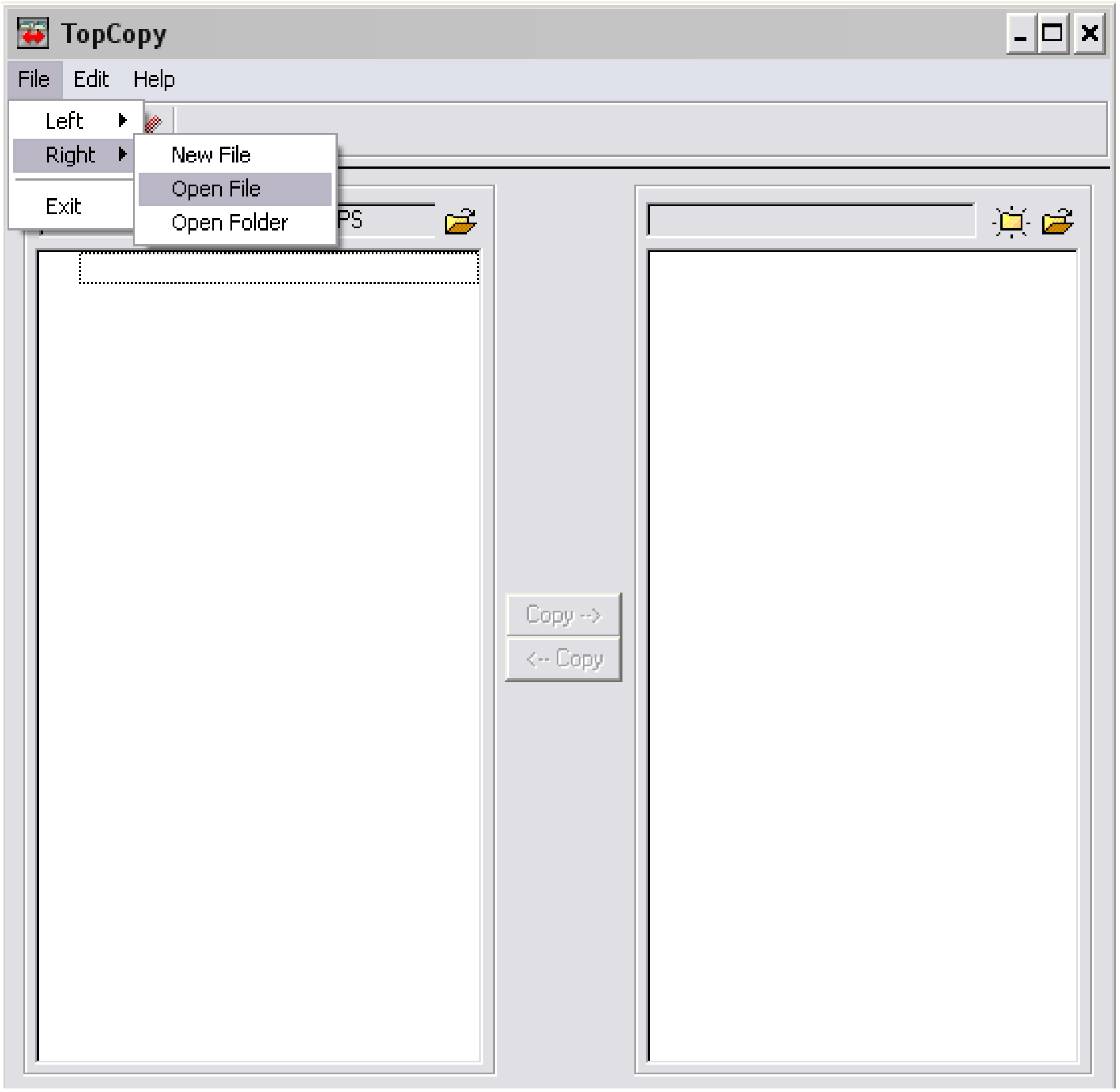


Figure 5. Preparing to import a file

Follow the Select File dialog to the source file.

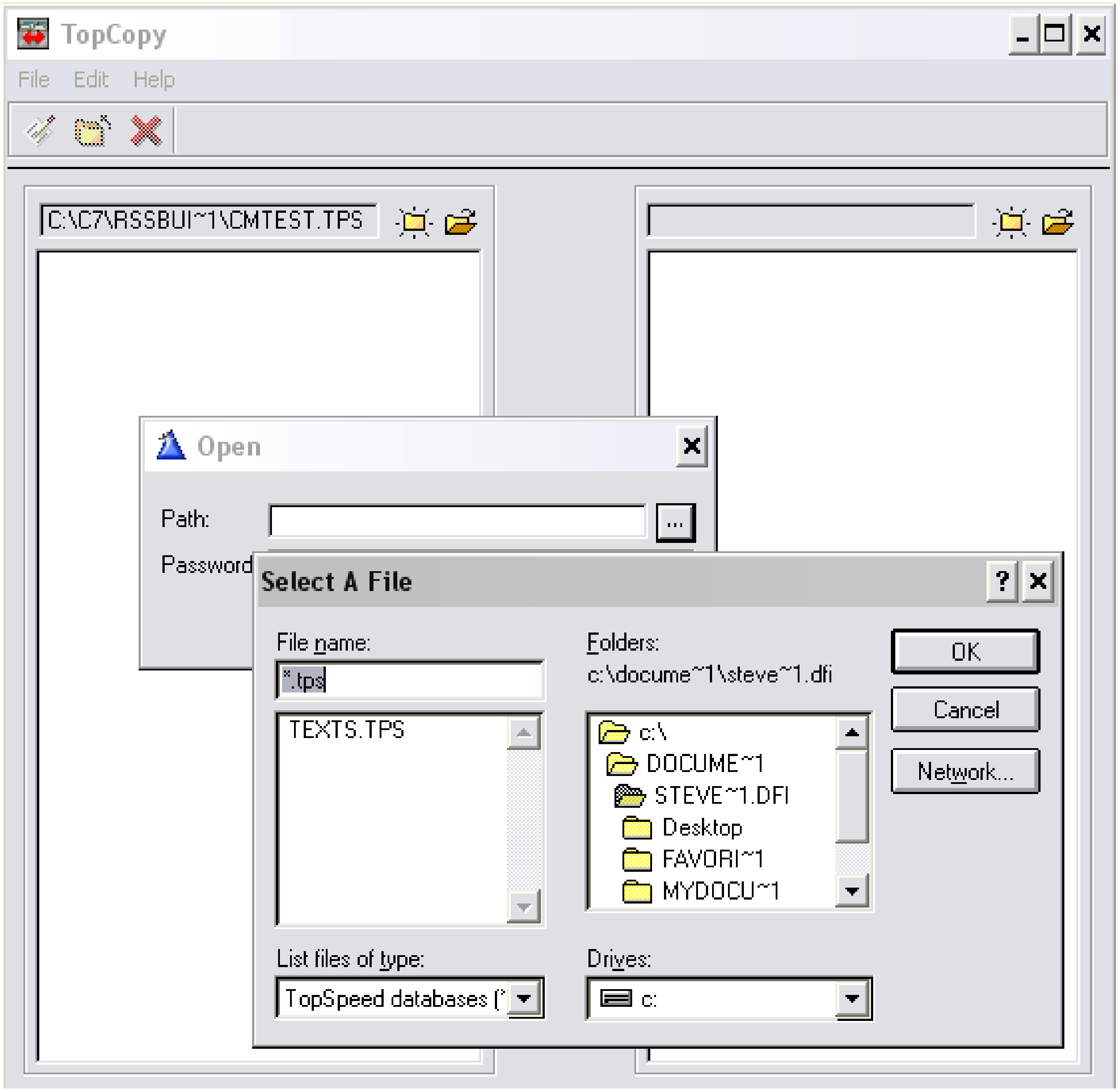


Figure 6. Select the file to import

Click the right-to-left copy button and the table (UNNAMED) appears in the new superfile.

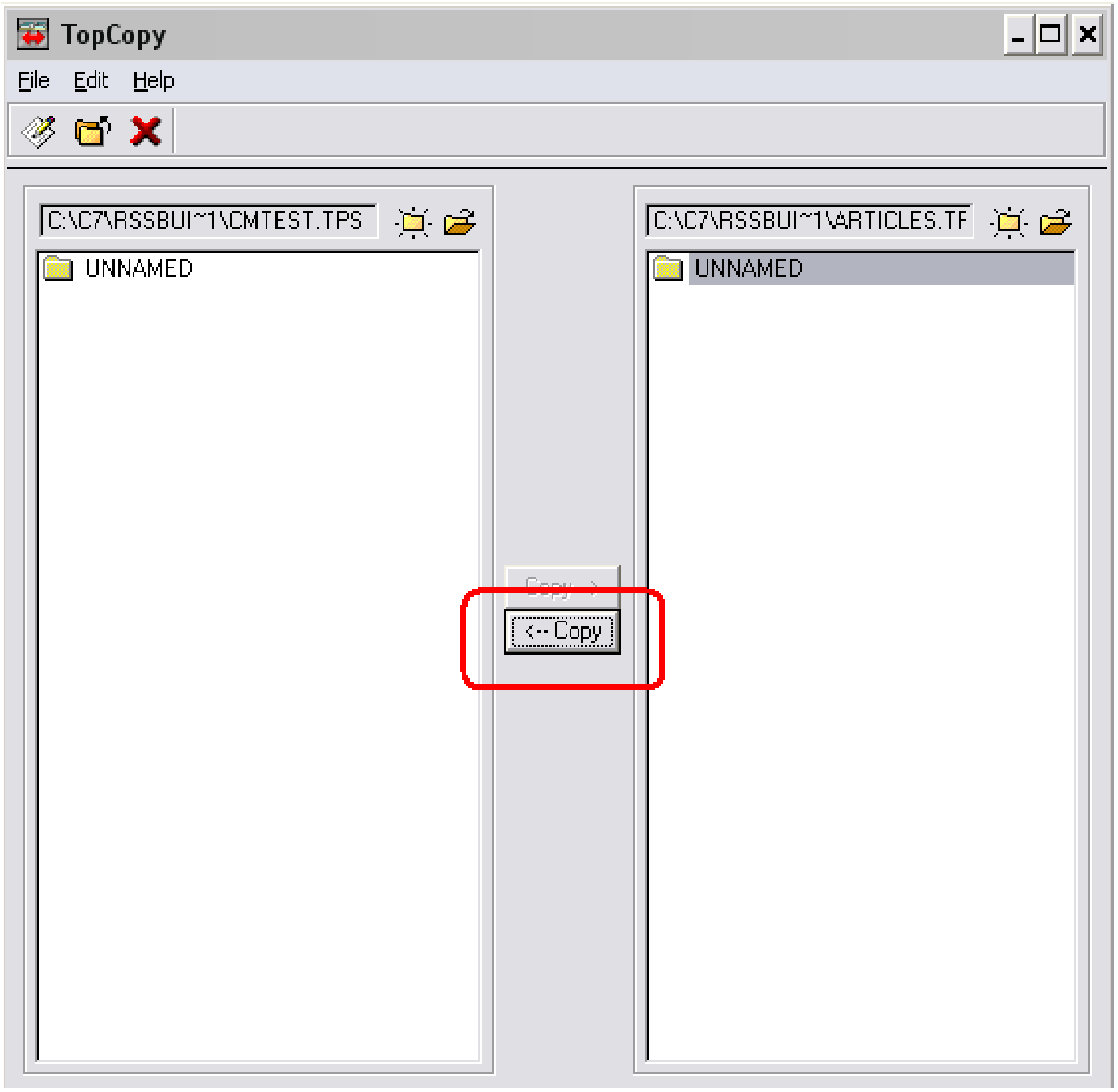


Figure 7. Copying the file to a table

Of course, "UNNAMED" won't quite do, will it?

Highlight "UNNAMED" in the left frame, click "Edit" in the top menu and select "Rename."

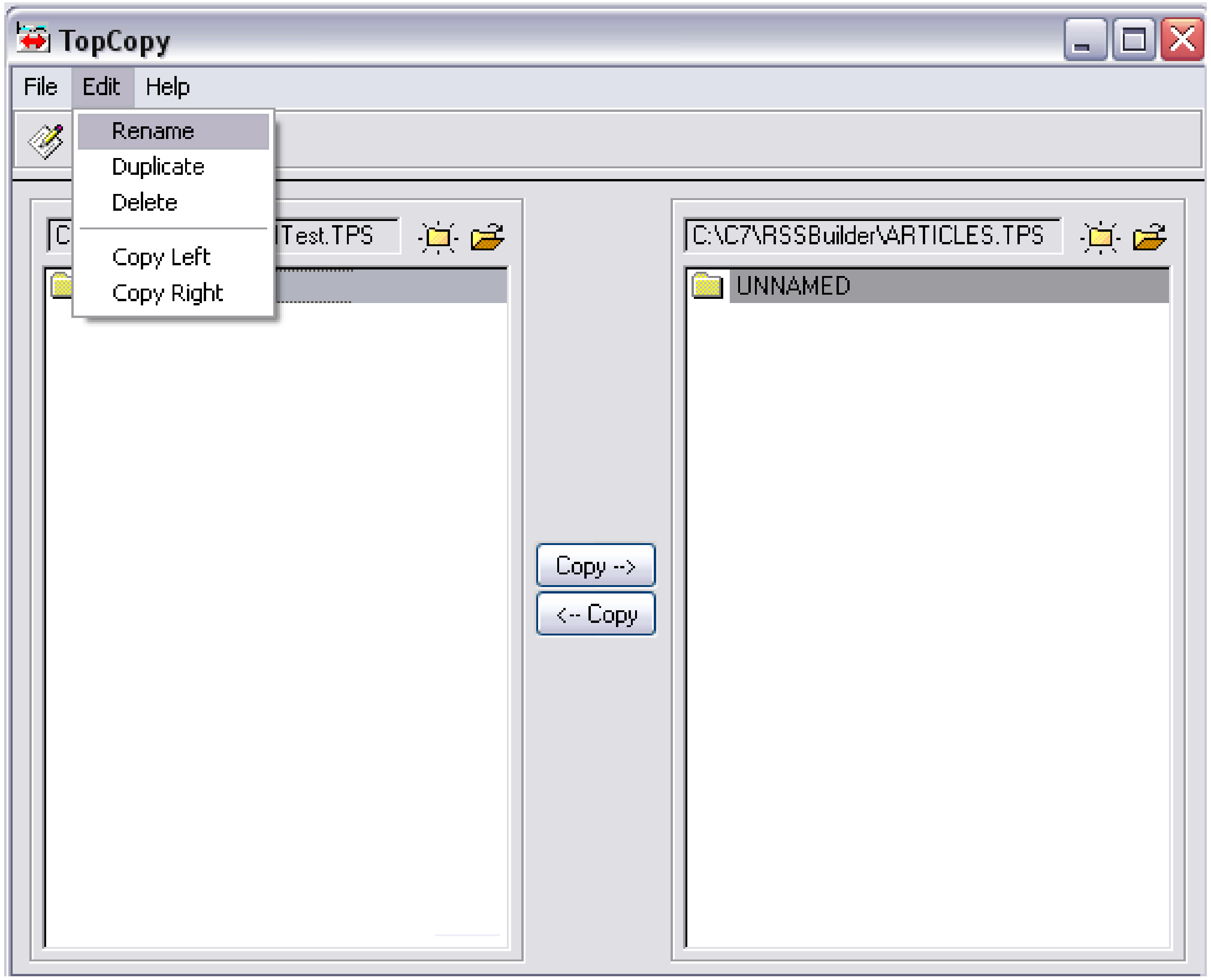


Figure 8. Preparing to (re)name the table

And, enter the proper name for the table:

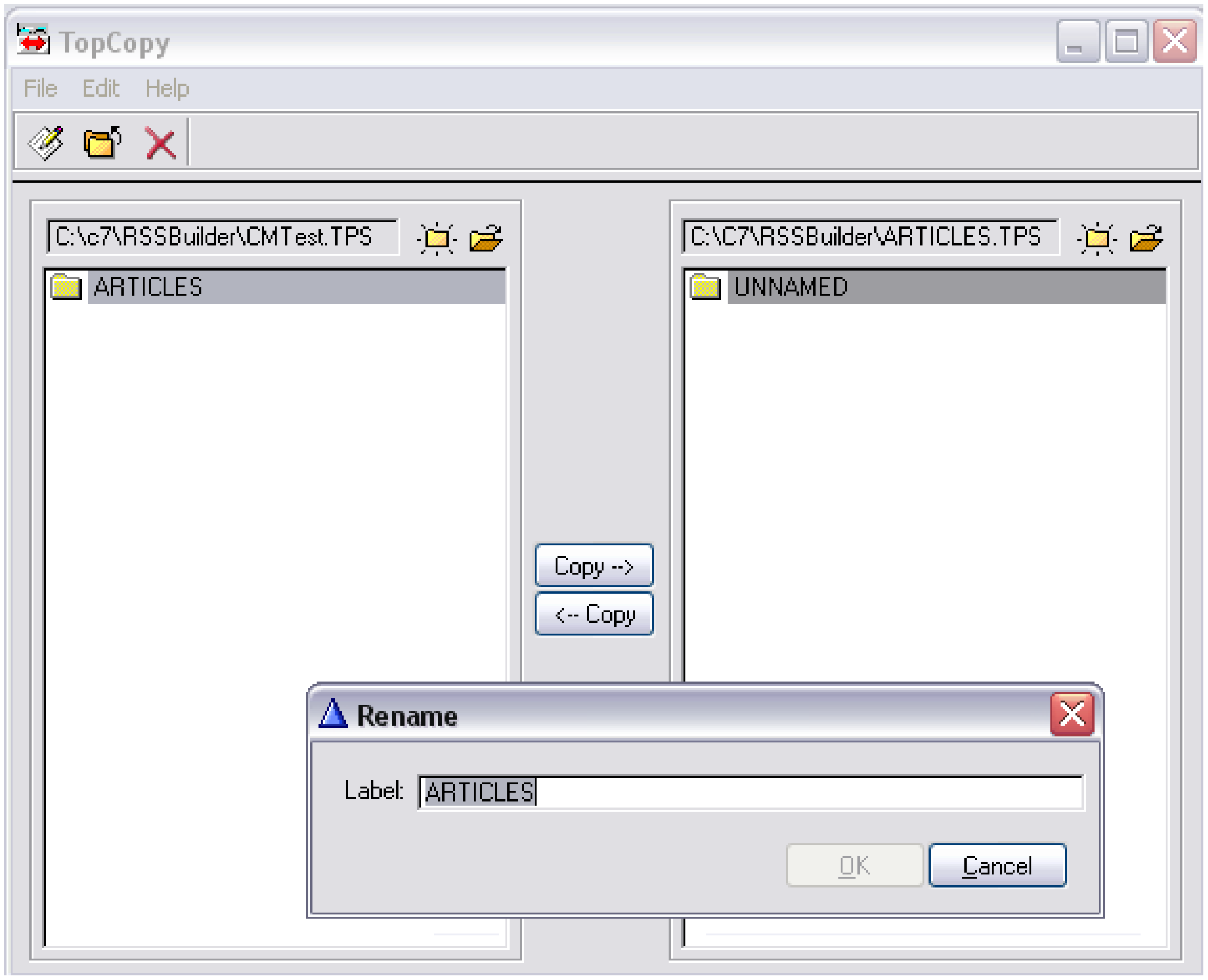


Figure 9. Give the table its proper name

If you do this for multiple tables and open the resulting TPS file using Topscan, the old files have been properly converted to internal tables. The old physical files, however, are still there, on disk, untouched.

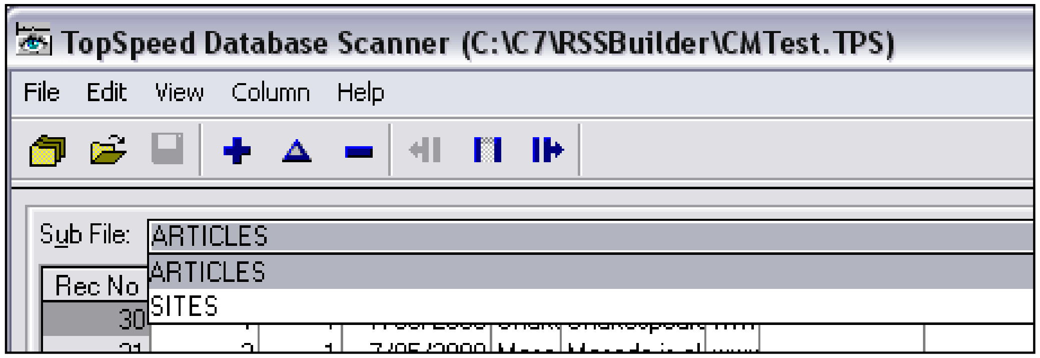


Figure 10. Resulting superfile with its tables

The fly in the ointment? I still have to update the dictionary for my app to reflect the fact that the files now reside in superfiles. Well, there's nothing to automate that, is there?

This provides another reason for always using a Name variable in the dictionary and, as I discussed years ago (Exploring the NAME attribute), creating the variable in the dictionary with a default value. Conversion to superfiles would require only changing that default value. In cases where I set the variables' values on app open (so that I can prepend a data path, for example), I may need to change the setup procedure.

Of course, I do need to recompile.

But wait!

"How can you call this 'something better'?" If anything, it looks like more work (especially considering the number of mouse clicks and my advancing carpel tunnel).

True, so far as it goes. But.

But, Topcopy has a command line interface. Therefore, it is entirely possible to create a small program that repeatedly RUNs that command line interface (of course, with appropriate parameter changes). I've included the important parts of the documentation at the end of this article (check, especially, the last example).

Summary

Topcopy makes converting your app, in whole or in part, to superfiles pretty easy.

Like my DOS conversion program, I can create an interface that asks for the data path.

Topcopy is also the answer to "How can a extract a table from a superfile into a standalone file?"

I recommend checking the Topcopy documentation.

Topcopy Command Line Usage

The database file should NOT be open when running the utility. Ensure that the file is closed before starting the utility. If exclusive access is denied, the utility displays an "Access Denied" warning.

To prevent access until the copy process is completed, the utility locks the file automatically.

TOPCOPY sourcefile[?password]!tablename destinationfile[?password]!tablename [/R] [/D]

TOPCOPY	The executable (TOPCOPY.EXE)
sourcefile	The DOS filename for the source TopSpeed database file from which tables are copied
?password	The source file's password (OWNER attribute). If the file has a password, it must be supplied. If the source database has no password, this can be omitted.
!tablename	The name of table within the TopSpeed Database from which to copy. If the database contains only one table, it is named UNNAMED.
destinationfile	The DOS filename for the target TopSpeed database file to which a table is copied. Type in a filename (including an extension) to create a new file.
?password	The destination file's password (OWNER attribute). If the file has a password, it must be supplied. If the destination file is being created, it will use this password.
!tablename	The name of the Table within the database to which the source file is copied. This defaults to the table name from the source database. If you want to create a single-table database, this should be named UNNAMED.
/R	Specify this switch if you want to remove the table from the source database after it is copied.
/D	Specify this switch to delete a source database that contains no remaining tables (remove the physical file from disk).

/K	Do a full BUILD of the source database keys prior to copying.

Example:

TOPCOPY orders.tps?acme!CUSTOMER customer.tps?acme96!UNNAMED

Copies the customer table from ORDERS.TPS (using the password--acme) to a single-table file CUSTOMER.TPS which has the password-acme96.

TOPCOPY orders.tps?fred!CUSTOMER customer.tps?barney!UNNAMED

Copies the customer table from ORDERS.TPS (using the password--fred) to a single-table file CUSTOMER.TPS which has the password-barney.

TOPCOPY customer.tps!UNNAMED orders.tps?acme96!CUSTOMER

Copies the UNNAMED (the only table in the file) table from CUSTOMER.TPS (which has no password) to the CUSTOMER table in ORDERS.TPS which has the password-acme96.

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

Reader Comments

Add a comment

Clarion Magazine

AES Encryption And Test Vectors

by Dave Harms

Published 2009-09-30

A few years ago Carl Barnes wrote a couple of articles for Clarion Magazine on compiling C code in Clarion, and he included as an example some C code that implements AES, the Advanced Encryption Standard. AES is one of the most popular and secure encryption standards, and was adopted by the US government in 2002.

Rakesh Khatri came across those articles and wanted to know how to implement test vectors such as those listed in the Wikipedia entry.
Test vectors are sets of known inputs and keys that generate known encrypted results (ciphers).

Inputs, keys and outputs are not necessarily text characters. Accordingly, the test vectors are all presented, in the article, as double-byte characters, in other words as 16 bits per character. Here's a sample input from the Wikipedia article:

"4EC137A426DABF8AA0BEB8BC0C2B89D6"

To translate this into a Clarion string you split the characters up into pairs:

4E C1 37 A4 26 DA BF 8A A0 BE B8 BC 0C 2B 89 D6

and then treat each as a hex character code. Keep in mind that all hex numbers must begin with a number and end with h, so if the first character is a letter you must prefix it with 0.

```
'<4Eh><0C1h><37h><0A4h> ... and so on
```

Here's a version of the code that works (with line breaks added). Note that I'm using SetKeyBits_Low instead of SetKeyBits_High. To get a test vector to pass you also need to the algorithm to use 128 bit encryption (SetKeyBits_Low) rather than the 256 bit encryption in Carl's original example.

```
  program

  include 'aes_h_inc.inc'



EnKey       String('<00h><01h><02h><03h><05h><06h><07h>↵
             <08h><0Ah><0Bh><0Ch><0Dh><0Fh><010h>↵
             <011h><012h>')
```

```
ctx              LIKE(aes_context) ! AES class
ret              signed

encrypt_16In     STRING(16)
encrypt_16Out    STRING(16)
decrypt_16In     STRING(16)
decrypt_16Out    STRING(16)
expected         string('<0D8h><0F5h><032h><053h><082h><089h>↵
                   <0EFh><07Dh><06h><0B5h><06h><0A4h>↵
                   <0FDh><05Bh><0E9h><0C9h>')

a                string('<11>')
b                string('<0Bh>')


   CODE

   ret = aes_set_key(ctx, EnKey, SetKeyBits_Low )
   if ret<>0 then Message('aes_set_key failed ret=' & ret).



   encrypt_16In = '<050h><068h><012h><0A4h><05Fh><08h><0C8h>↵
                   <089h><0B9h><07Fh><059h><080h><03h>↵
                   <08Bh><083h><059h>'
   aes_encrypt( ctx, encrypt_16In, encrypt_16Out )

   if encrypt_16Out = expected
      message('success!')
   ELSE
      message('failure')
   END
```

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

STARTing procedures by address

Published 2009-09-01

Blogs on this site

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

In the SV newsgroups James Hrubes asked a question about whether it's possible to START a procedure via the procedure's name rather than its label. The short answer is no. Say you have a procedure called BrowseStudents. BrowseStudents is a label, and 'BrowseStudents' is a string, and the Clarion runtime library doesn't provide a way of translating a string to a procedure label.

In response to James' question, Maarten Veenstra posted this little gem from an unknown author. This program illustrates another approach, which is to store the ADDRESS of a procedure and then START the address instead of the label.

But there's a trick. Take a close look at the DynaStart prototype in the MAP:

```
   Program
include('builtins.clw')
Procs  LONG,DIM(16)

 MAP
  Main
  BrowseStudents
  BrowseTeachers
  BrowseSchools
  BrowseClassRooms
  MODULE('Clarion')
    DynaStart(LONG,LONG),LONG,PROC,NAME('Cla$START')
  END
 END

 CODE

 Procs[1] = address(BrowseStudents)
 Procs[2] = address(BrowseTeachers)
 Procs[3] = address(BrowseSchools)
 Procs[4] = address(BrowseClassRooms)
 MAIN

Main  PROCEDURE
Window WINDOW('Select Procedure'),AT(,,219,69),|
         FONT('MS Sans Serif',8,,FONT:regular,CHARSET:ANSI),|
         CENTER, GRAY
       LIST,AT(93,18,103,10),USE(?List1),VSCROLL,DROP(4),|
         FROM('BrowseStudents|BrowseTeachers|BrowseSchools|BrowseClassRooms')
       PROMPT('Select Procedure:'),AT(23,18),USE(?Prompt1)
       BUTTON('Start Selected Procedure'),AT(60,44,99,14),USE(?StartButton)
     END
     CODE
     OPEN(Window)
     SELECT(?List1,1)
     ACCEPT
     CASE FIELD()
      OF ?StartButton
       CASE EVENT()
       OF EVENT:Accepted
        DynaStart(Procs[CHOICE(?List1)],25000)

       END
     END
     END

BrowseStudents PROCEDURE
    CODE
     MESSAGE('Students')

BrowseTeachers PROCEDURE
   CODE
```

```
    MESSAGE('Teachers')

  BrowseSchools     PROCEDURE
    CODE
    MESSAGE('Schools')



  BrowseClassRooms  PROCEDURE
    CODE
    MESSAGE('ClassRooms')
```

You might think you could use START directly, but you can't. The libsrc\builtins.clw contains the following prototype:

```
   START(_PROC,UNSIGNED=0),SIGNED,PROC,NAME('Cla$START')
```

And _PROC is a procedure prototype:

```
  _PROC(),TYPE
```

The compiler complains if you attempt to pass an address instead of an actual procedure.

As the unknown author of that example deduced, internally START simply takes the address of a procedure. So the redeclaration of START uses LONG parameters instead, and now you can pass the address of any parameterless procedure. Presumably you could take the same approach with the other two forms of START.

It's a nifty bit of code. So who wrote it?