# Clarion Magazine

## Clarion News

- » J-Skype Released
- » CPCS Report Design and Consulting Services
- » SetupBuilder 5.6 Build 1636
- » New Host For Clarion Connection
- » xPathManager 1.4.C63
- » Snazzy Listbox 1.26
- » xFText 2.7
- » Free Wallpaper and Icons
- » SetupBuilder 5.6 Professional Edition Released

[More news]

## Latest Free Content

- » SV Releases Clarion Roadmap
- » OSCON: Query Interfaces, PHP/Web 2.0, and TimeTravel Tables
- » OSCON: solidDB Beefs Up MySQL OLTP Choices
- » OSCON: Wednesday Keynotes
- » Clarion Tips & Techniques, Volume 3 Available Now

[More free articles]

## Clarion Sites

- » Ingasoftplus
- » RADFusion

- » PINVOKE.NET
- » SealSoft Company
- » Lodestar Software (CWaddons)
- » The Clarion Connection
- » BoxSoft (a.k.a. Mike Hanson)
- » CapeSoft
- » SoftVelocity Community Site
- » SoftVelocity Company Site

[More sites]

## Podcast

SoftVelocity president Bob Zaunere is back on Planet Clarion. Andrew and Dave talk with Bob about the ConDev AVIs and what they really mean, what's involved in creating the new IDE, and the differences between Win32 and .NET screen formatters. 47:00:00, 16913K

Listen now

[Track lists, more podcasts]

## Clarion Blogs

- » DevDawn
- » John Griffiths
- » Knobblegrud
- » JohnMo
- » SoftVelocity
- » Lesley Dean
- » DevDawn
- » Gary James

- » Clarion Magazine

"One of my most valuable tools."
▸ Shawn Simmons

CLARION MAGAZINE

## Latest Subscriber Content

### ClarionMag Price Increase Oct 15

As of October 15, 2006 Clarion Magazine subscription prices will be increasing by $20 due to continuing unfavorable foreign exchange rates (we charge in US dollars but are located in Canada).

Posted Tuesday, October 03, 2006

### Clarion 101: Understanding the IDE

Dave Harms kicks off ClarionMag's new Clarion 101 series with an overview of the AppGen, templates and dictionary editor. Also included: links to further reading.

Posted Thursday, September 28, 2006

### Solving Sudoku Puzzles With Clarion, Part 2

Jon Waterhouse explains his Sudoku solver, written in Clarion. This program uses a deductive approach as much as possible, and falls back on guessing only where deduction is impossible. Part 2 of 2.

Posted Wednesday, September 27, 2006

### Solving Sudoku Puzzles With Clarion, Part 1

Jon Waterhouse explains his Sudoku solver, written in Clarion. This program uses a deductive approach as much as possible, and falls back on guessing only where deduction is impossible. Part 1 of 2.

Posted Thursday, September 21, 2006

### Roll Your Own Web Search Engine

Jim Albrecht shows how you can create your own web search engine in Clarion using Skip Williams' WebFetch DLL.

Posted Thursday, September 14, 2006

### Storing GROUPs in INI Files

Jeff Slarve demonstrates a small function that stores any group as field/value pairs in an INI file, and can also restore data from an INI file.

Posted Friday, September 08, 2006

### A Template Debugger

Building on Mark Goldberg's technique, Russ Eggen shows how easy it is to add real-time debug logging to any template.
Posted Wednesday, September 06, 2006

### An Economical Record Status Control

Nardus Swanevelder condenses six controls and their prompts into a single line control displaying "user created/changed" status information.
Posted Monday, August 28, 2006

### Commentary on UKCUG, Clarion Roadmap

Dave Harms comments on some additional screen shots from the UKCUG meeting, and SoftVelocity's newly published roadmap for Clarion.
Posted Monday, August 21, 2006

### SV Releases Clarion Roadmap

Bob Zaunere has released SoftVelocity's roadmap for Clarion through 2007, with details on the Win32/Win64 versions and Clarion.NET for .NET 2.0 and 3.0.
Posted Thursday, August 10, 2006

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

## Printed Books & E-Books

### E-Books

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

### Printed Books

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

## 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

## Gitano Distributors Wanted

Gitano is looking for distributors for its Clarion utilities and icon sets. Contact Jesus Moreno at mail @ gitanosoftware dot com.
Posted Tuesday, October 03, 2006

## Ingasoftplus Forum Returns

The Ingasoftplus product support forum is back online.
Posted Tuesday, October 03, 2006

## J-Skype News

Gary James has announced a number of new J-Skype features, including: A Skype robot (to test send a message to Skype ID jskype_test); Set and retrieve the online status, mood text, profile information, etc; Use Skype for inter-application communication (handy since Skype can get through most firewalls). Price increase (from $49) is coming soon.
Posted Tuesday, October 03, 2006

## Webmaster Icon Collection

Webmaster is a collection of tags, seals and buttons for your projects. Although called the 'Webmaster' collection, these images can be used in any projects where you want to emphasize a special sale, guarantee, purchase, new, etc. Number of icons: 355 (947meg, 11,360 individual files). Formats: ico, .psd, .png, .gif, .bmp, .jpg, icns. Sizes: 64x64, 72x72, 96x96, 128x128, 256x256

Posted Tuesday, October 03, 2006

## Virtual and Dedicated Server Sale for Clarion Developers

Oak Park Solutions has a special on for Clarion developers and their clients. Virtual Dedicated Server Plans From $34.95/month, featuring up to 50 GB disk space, up to 2,000 GB bandwidth, up to 3 dedicated IP addresses, and full admin access. Dedicated Servers also available. No long-term commitments required. Hosting accounts are also available from $4.99 a month with no long-term commitment.
Posted Tuesday, October 03, 2006

## Whitemarsh Newsletter for September 2006

The Data Interoperability Community of Interest Handbook is finished. The table of contents and the first chapter are posted for review. A pre-publication sale announcement will occur in two weeks. The second item is a paper that Mike Gorman co-authored with Dr. Andreas Tolk of Old Dominion University, and Leslie Winters of the DoD's Joint Forces Command. This paper: "Next Generation Data Interoperability, It's All About Metadata" was delivered to the September 2006 Fall Simulation Interoperability Workshop, in Orlando, FL. Whitemarsh is going to incorporate these levels into the Data Interoperability Community of Interest Handbook. The third item is the latest edition to the Whitemarsh Short Paper Series, Database Schema Management. This paper focuses on the creation of enterprise-wide semantics from existing database schemas. Explained is the overall process, the creation of a layer of common semantics, the data model templates, and the enterprise-wide data element semantics. References are also provided to a key document in the Whitemarsh website called "Reverse and Forward Engineering".
Posted Tuesday, October 03, 2006

## J-Skype Released

Strategy Online has released J-Skype, a global extension template that makes it easy to call the Skype API. You can send messages to other Skype users, make Skype calls, and more. You can also call Skype across threads, and all message handling is processed by the class. Now $49, including a free upgrade to the more expensive Gold version. Demo available.
Posted Thursday, September 28, 2006

## CPCS Report Design and Consulting Services

CPCS has a new Report Design and Consulting Service which provides design, development, and consulting aimed at helping Clarion developers with the reporting process. CPCS has long been the leader in reporting based addons for Clarion. Now you can outsource some or all of your reporting needs to CPCS.
Posted Wednesday, September 27, 2006

## SetupBuilder 5.6 Build 1636

SetupBuilder 5.6 Build 1636 is a very important update which resolves the Data Execution Prevention (DEP) issue. DEP security updates in Windows XP SP2, Windows x64, and Windows 2003 stopped SetupBuilder 5.x executables.
Posted Wednesday, September 27, 2006

## New Host For Clarion Connection

The Clarion Connection has moved; until the redirection kicks in you can use this link.
Posted Wednesday, September 27, 2006

## xPathManager 1.4.C63

xPathManager lets your program work with data in different folders. Features include: Store paths in your own table; Customize windows to select and edit paths; Full or relative paths; Create missing path; Save last opened path; Map/unmap network drives.
Posted Wednesday, September 27, 2006

## Snazzy Listbox 1.26

In Snazzy Listbox 1.26 the "fixed" column attribute now works. Available at Steve Parker's Download Center.
Posted Wednesday, September 27, 2006

## xFText 2.7

xFText lets you place any text and bitmap images anywhere in the application frame. No DLL - just Clarion source and Windows API calls. Supports SingleExe, MultiDll (Local Mode, Standalone Mode), 32-bit. Compatible with Clarion 6.x (6.3, 6.2, 6.1), Clarion 5.5, Clarion 5. Features include: Set global text margins; Set all font attributes; Set normal, light and dark text color for 3D text animation; Add, change, and remove frame text at runtime; Execute code on mouse click; Set parameters by variables.

Posted Wednesday, September 27, 2006

## Free Wallpaper and Icons

Gitano has release some free Clarion wallpaper and an icon collection. In the Products menu select FREE Products.
Posted Wednesday, September 27, 2006

## SetupBuilder 5.6 Professional Edition Released

SetupBuilder 5.6 Professional Edition starts at $179 USD for a royalty-free usage license. A trial version is available.
Posted Wednesday, September 27, 2006

## SetupBuilder 5.6 Developer Edition Released

SetupBuilder 5.6 Developer Edition starts at $299 USD for a royalty-free usage license. A trial version is available.
Posted Wednesday, September 27, 2006

## PrintWindow 1.05

New in PrintWindow 1.05: Ampersand (&) no longer stripped from "string" type controls; RTF controls without "USE" variable were causing compilation errors in C6; From this version on, your installer settings will be memorized in the registry.
Posted Wednesday, September 27, 2006

## Jiff On Leave

Capesoft's Geoff Thomson is on leave and will be back October 2. Please mail support at capesoft dot com directly for support.
Posted Wednesday, September 27, 2006

## NetTalk Web Server Example Updated

The latest live NetTalk web example, using build 4.16, is now online. While still not a "complete app" it does show off some of the new features in this release, including: Update an invoice, Customer field, type in a customer code and note what happens. (valid codes are like 1 thru 10 or so); Update an invoice, Customer field, type in a customer

_name_ and note what happens. (valid names are Iskor, Google, Microsoft and some others); Browse Customers / Change (Iskor) / Invoices Tab. / Select the Paid / Unpaid & Both radio buttons in turn; Browse Customers / Change / General Tab / Select Country (from the drop down) and notice the list of available Shippers changes (try Asutralia, UK, USA or South Africa)
Posted Friday, September 15, 2006

## NetTalk 4.16

The latest beta build of NetTalk (4.16) is now available. Highlights include: Fully interactive forms - do anything you want whenever a user completes a field; Examples of reports with "option screens"; File uploading on forms (still primitive, but it's there); Graphical buttons (use FF for best results); Support for "smart" Lookup fields (enter a code _or_ a description).
Posted Friday, September 15, 2006

## iQ-XML 1.22

iQ-XML version 1.22, compiled for build C6.3.9054, is now available. Changes include: Writer is now up to 50% faster; XML:CreateXMLString() could add unnecessary spaces before elements, fixed; XML:SetFieldPicture function was not threaded, fixed.
Posted Friday, September 15, 2006

## Clarion Developers Hosting Services

Oak Park Solutions has opened up its products to Clarion developers and their clients. Products incude domain names, business registrations, hosting plans, virtual dedicated servers, dedicated servers, fax through email, and more.
Posted Friday, September 15, 2006

## Translator Plus Version 63-05

Translator Plus Version 63-05 is now available for download. This reflects changes in Clarion 6.3 hotfix 9054. The modified Legacy (Clarion) templates have been updated to reflect the latest changes made by Softvelocity.
Posted Friday, September 15, 2006

## Clarion 6.3 9054 Released

Build 9054 is now available for download to registered Clarion 6.3 users. New features include EIP improvements and new RTF class methods. Numerous bug fixes.
Posted Wednesday, September 13, 2006

## Dr. Explain Promotion

During the fall almost everyone can receive a significant discount on
Posted Monday, September 11, 2006

## CPCS Support Availability

CPCS Support will be unavailable from 9/07/2006 thru 9/14/2006.
Posted Monday, September 11, 2006

## Fomin Tools Server Move

The Fomin Tools site has moved to another web hosting provider. The web site itself and all the online services, including support forum and live update, are online as of September 1, 2006.
Posted Monday, September 11, 2006

## SimFileLauncher Updated

A new version of SimFileLauncher is now available. This version has a search facility that allows you to search any folder recursively for all files matching the criteria you set. You can even search your whole hard drive if you want to. You can add file extensions (examples: .ico, .hlp, .htm, .chm) then when you do your search, check the ones you want included in the search; alternatively you can enter all or part of a file name you want to find. The recursion returns a list of files, and you then have the opportunity to select and launch one of them or add it to your existing database of files you regularly need to launch. The new version also also allows you to elect to have it remain on the taskbar when you close SimFileLauncher. SimFileLauncher, which can be accessed through the Clarion IDE, lets you reduce the number of help files in the IDE and simply access them through SimFileLauncher. The price remains at $29 US, and the upgrade is free to existing users. You can download and use SimFileLauncher for 30 days before you need to purchase it.
Posted Monday, September 11, 2006

## Gitano Software New Look and Policy Changes

Gitano's web site has been redesigned. The following are changes to the site and policies: New price lineup - a new lower pricing is in place for all utilities, regular and with source; Due to the email issues that continue to arise all support will now only be handled through the support form and the soon to be available open support forum.
Posted Monday, September 11, 2006

## SetupBuilder 5.5 Developer Edition HotFix

If you have SetupBuilder 5.5 Developer Edition (build #1589) with an expired maintenance subscription and you decided not to renew your subscription, the UPDATEprotect feature disabled itself. A hotfix patch to fix this problem is now available. This patch is *not* required if: You have a valid and active maintenance subscription; you are not on build #1589; You are not using the UPDATEprotect subscription feature. The patch does not increase the build number nor does it include other
Posted Monday, September 11, 2006

Clarion Magazine

# ClarionMag Price Increase Oct 15

Published 2006-10-03

As of October 15, 2006, Clarion Magazine's prices will go up as follows:

| **Product** | **Old price** | **New price** |
|---|---|---|
| One Year Subscription (includes all back issues) | $149 | $169 |
| One Year Renewal (current subscription) | $99 | $109 |
| Two Year Subscription (includes all back issues) | $229 | $249 |
| Two Year Renewal (current subscription) | $179 | $199 |
| E-Books - regular price (each) | $19.95 | $24.95 |
| E-Books - subscriber price (each) | $9.95 | $11.95 |

Although we charge in US dollars, we're located in Canada. Continuing unfavorable exchange rates make it necessary for us to raise our prices.

**Current subscribers:** View your subscription expiry date here.

## Reader Comments

Add a comment

Clarion Magazine

# Clarion 101: Understanding the IDE

## by Dave Harms

Published 2006-09-28

*This article is the first in Clarion Magazine's Clarion 101 series.*

If, like me, you've been using Clarion for many years, the Clarion way of doing this is second nature (well, hopefully). But if you're new to Clarion, or if you're trying to explain Clarion to another developer, the Clarion concept may seem like the weirdest thing since Microsoft Bob.

Microsoft Bob is long gone (if you're lucky, you might find a copy on eBay for, say, $15), but after a quarter of a century Clarion is still here, and still making money for Clarion developers. Why? In short, because we're all too old to switch to another language! Ha ha. Actually I know a whole lot of Clarion developers who use other languages along with Clarion (I'm one of them), and I keep hearing from Clarion developers who leave for another language, and then come back.

Why do Clarion developers stick with Clarion, and come back to Clarion? One word: Productivity. Even with its 1990s style (and soon to be replaced) IDE, you can build useful business apps with Clarion in less time, with less work, than with any other IDE I know of.

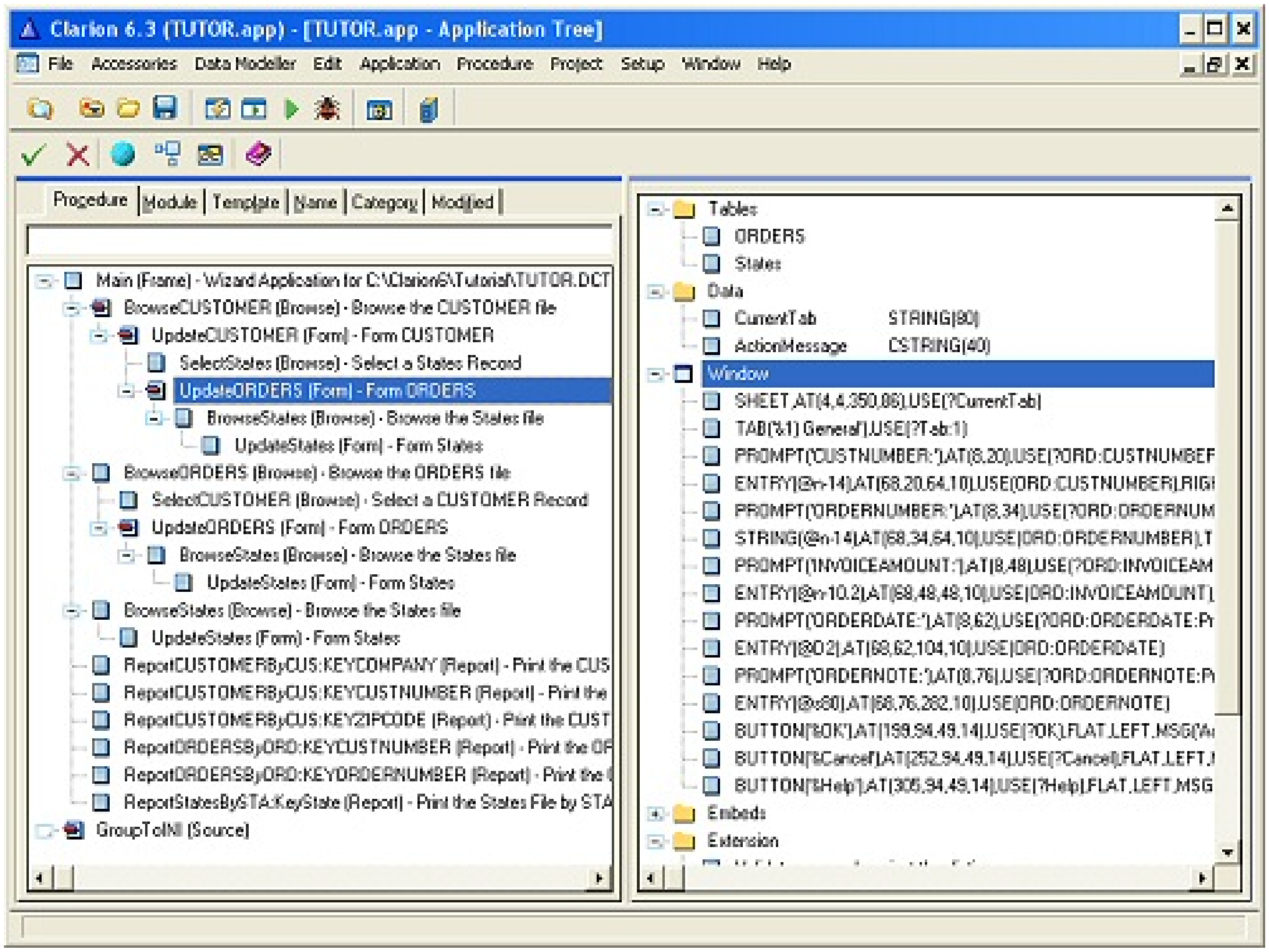
In this article and the next I'll review the fundamentals of the Clarion development system; in later articles in the 101 series I will expand on each of these areas, and provide links to other Clarion Magazine articles where you can mine all kinds of programming gold.

## The Clarion IDE (up to C6)

Soon to be replaced by Clarion 7, the Clarion 6 IDE is the last of its line. Yes, it's a yellow-toothed, hoary 16 bit beast and its days are numbered. But it's also a beast of burden – you can hitch an incredible programming load to C6. I'm constantly amazed at the size and complexity of the applications Clarion developers routinely create. We seem to think nothing of data dictionaries with hundreds of tables, of scores of controls on an update form, of multi-DLL applications with sometimes thousands of procedures, of interchangeable data drivers and complex reports. And that's only the beginning of what C6 does and can do.

And how does the IDE make developers so productive? By using the Application Generator to create the lion's share of your application's code using a combination of customizable templates, a data dictionary (in which you define your files/tables), and your own embedded code (if necessary, and for all but the simplest apps at least some embedded code is necessary). Clarion can also automatically create a working application from your data dictionary, if you wish, by using its Wizard templates. I'll talk about all of these features in more detail a little later.

The IDE for C6 and earlier versions is of a fairly typical multi-paned configuration. In application view you have a list of procedures (in various views) in the left pane, with some information for the selected procedure in the right pane. You can resize the panes but you cannot drag them around.


Clarion 6.3 (TUTOR.app) - [TUTOR.app - Application Tree]
File Accessories Data Modeller Edit Application Procedure Project Setup Window Help
Procedure Module Template Name Category Modified

```
Main (Frame) - Wizard Application for C:\Clarion6\Tutorial\TUTOR.DCT
  BrowseCUSTOMER (Browse) - Browse the CUSTOMER file
    UpdateCUSTOMER (Form) - Form CUSTOMER
      SelectStates (Browse) - Select a States Record
      UpdateORDERS (Form) - Form ORDERS
        BrowseStates (Browse) - Browse the States file
          UpdateStates (Form) - Form States
  BrowseORDERS (Browse) - Browse the ORDERS file
    SelectCUSTOMER (Browse) - Select a CUSTOMER Record
    UpdateORDERS (Form) - Form ORDERS
      BrowseStates (Browse) - Browse the States file
        UpdateStates (Form) - Form States
  BrowseStates (Browse) - Browse the States file
    UpdateStates (Form) - Form States
  ReportCUSTOMERByCUS:KEYCOMPANY (Report) - Print the CUS
  ReportCUSTOMERByCUS:KEYCUSTNUMBER (Report) - Print the
  ReportCUSTOMERByCUS:KEYZIPCODE (Report) - Print the CUST
  ReportORDERSByORD:KEYCUSTNUMBER (Report) - Print the OR
  ReportORDERSByORD:KEYORDERNUMBER (Report) - Print the
  ReportStatesBySTA:KeyState (Report) - Print the States File by STA
GroupToINI (Source)
```

```
Tables
  ORDERS
  States
Data
  CurrentTab        STRING(80)
  ActionMessage     CSTRING(40)
Window
  SHEET,AT(4,4,350,86),USE(?CurrentTab)
  TAB('%1) General'),USE(?Tab:1)
  PROMPT('CUSTNUMBER:'),AT(8,20),USE(?ORD:CUSTNUMBEF
  ENTRY(@n-14),AT(68,20,64,10),USE(ORD:CUSTNUMBER),RIG
  PROMPT('ORDERNUMBER:'),AT(8,34),USE(?ORD:ORDERNUM
  STRING(@n-14),AT(68,34,64,10),USE(ORD:ORDERNUMBER),T
  PROMPT('INVOICEAMOUNT:'),AT(8,48),USE(?ORD:INVOICEAM
  ENTRY(@n-10.2),AT(68,48,48,10),USE(ORD:INVOICEAMOUNT),
  PROMPT('ORDERDATE:'),AT(8,62),USE(?ORD:ORDERDATE:P
  ENTRY(@D2),AT(68,62,104,10),USE(ORD:ORDERDATE)
  PROMPT('ORDERNOTE:'),AT(8,76),USE(?ORD:ORDERNOTE:P
  ENTRY(@s80),AT(68,76,282,10),USE(ORD:ORDERNOTE)
  BUTTON('&OK'),AT(199,94,49,14),USE(?OK),FLAT,LEFT,MSG('A
  BUTTON('&Cancel'),AT(252,94,49,14),USE(?Cancel),FLAT,LEFT,
  BUTTON('&Help'),AT(305,94,49,14),USE(?Help),FLAT,LEFT,MSG
Embeds
Extension
```

**Figure 1. The Clarion 6 IDE with open application**

In dictionary view, the IDE is similarly configured, but with tables in the left pane, columns in the upper right pane, and relationships and keys/indexes in the lower right pane.

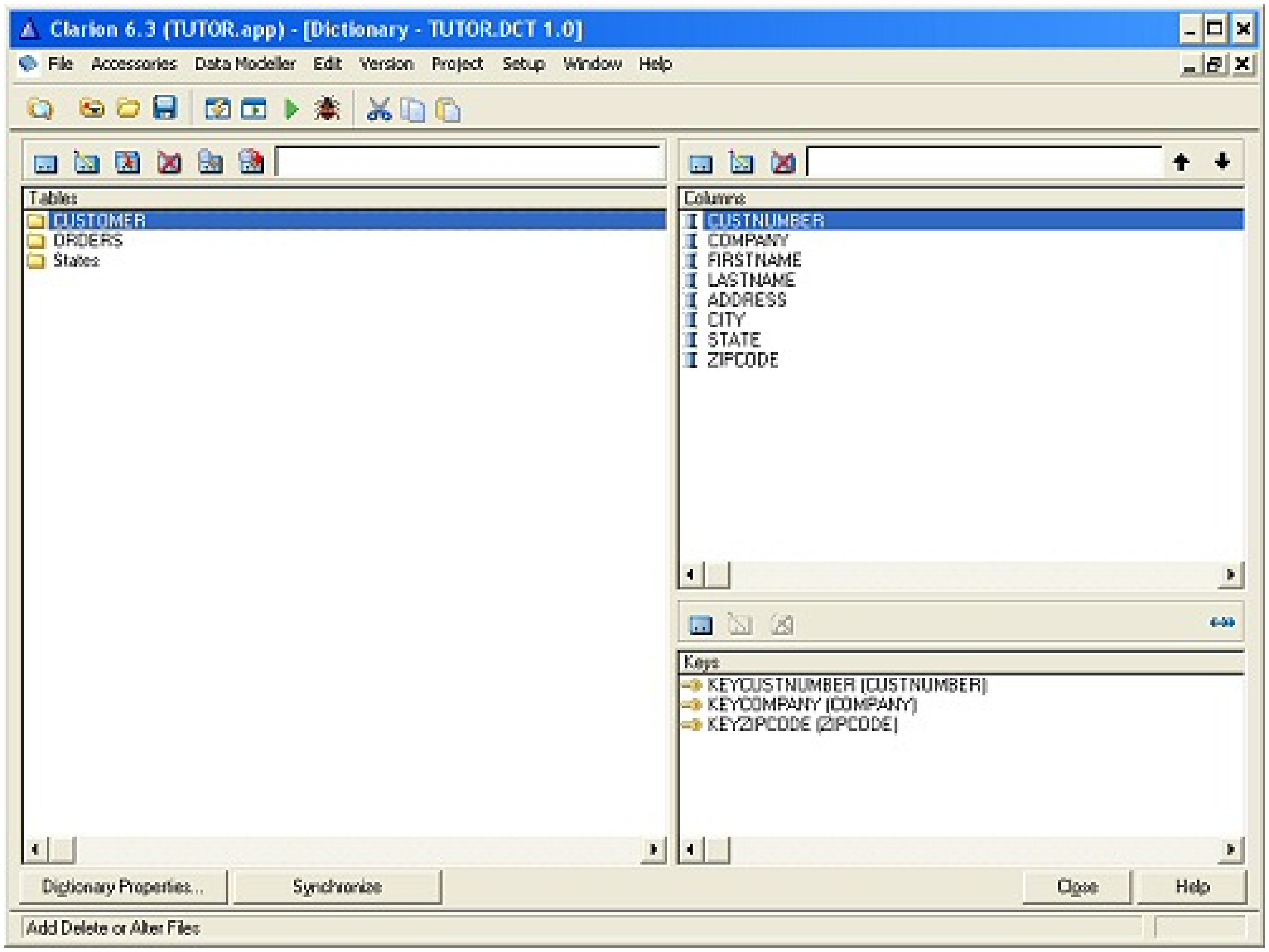


**Figure 2. The Clarion 6 IDE with open dictionary**

Admittedly, the current Clarion IDE is a bit on the clunky side. For one thing, you cannot work on the data dictionary and the application at the same time. Let's say you're busy modifying your application when you realize you need an additional field (column) in one of your files (tables). You have to close the APP file, open the DCT, make your changes, close the DCT, open your APP, and wait briefly (or not so briefly, depending on the size of your app and the speed of your processor) for the AppGen to merge the changes before you can continue working on the application. (You can open the dictionary in read-only mode while working on an APP, but not once you've started drilling down into template prompts - you can, however, use Application|View Dictionary, press the pushpin in the lower right hand corner, and keep a dictionary view window open at all times.)

The same time-waster holds for the templates which generate most (rarely all) of your code. It's unlikely you'll change the shipping templates, but you can create your own templates, and many programmers do this to great advantage. If you change a template, you need to re-open your application so the template modifications can be re-registered and incorporated.

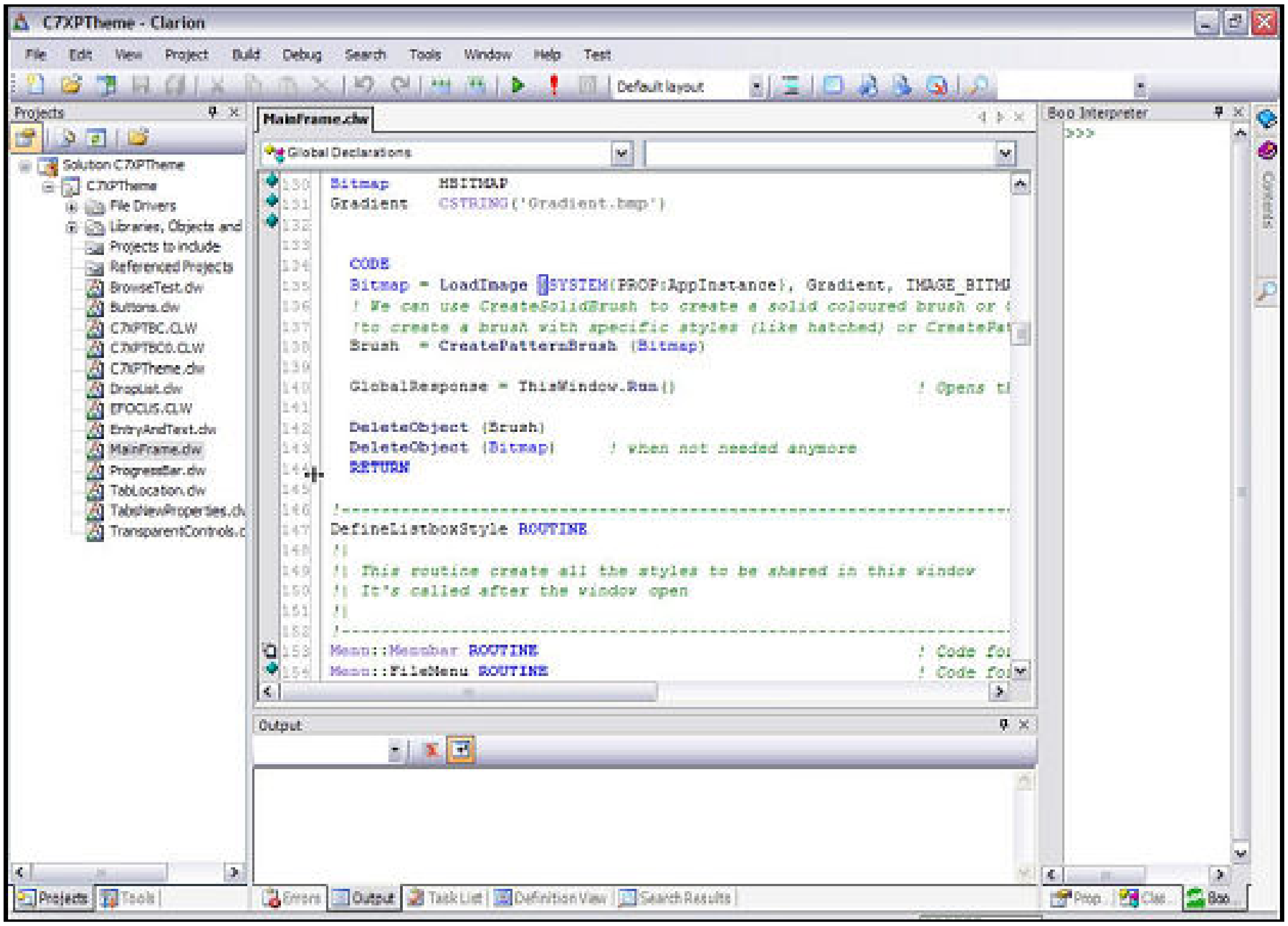
The other major issue with the current IDE is the "modal" nature of the interface. You often have to drill down through multiple windows of template prompts to change a setting or embed some code. That takes time.

These issues will be addressed to some degree (and perhaps solved) in Clarion 7. Yet despite these limitations, and the consequent whining and grumbling from the ranks, Clarion is still a massively productive application development system.

## The Clarion 7 IDE

As I’m writing this article, the C7 IDE has been shown only to a select few, and the extent to which it will change a Clarion developer’s workflow isn’t completely clear, although big productivity gains are certainly in the offing. SoftVelocity has licensed the source code for the very cool SharpDevelop IDE, which is much like Visual Studio, and is wiring (or has wired) #develop up to the Clarion IDE applets that make Clarion what it is. In other words, C7 will be a pretty new face on proven template-based code generation technology (and the platform for Clarion.NET, but that’s another set of articles). The main workflow improvement, and it’s a biggie, should come from the flattening of the IDE, which among other things means not having to drill down through a bunch of windows to make a template change or embed some code. You’ll also be able to arrange the various panels in the IDE as you wish.

There are other important changes coming in C7. In advance of the actual release you may want to read about the UK CUG presentation as well as the C7 video.



```
Bitmap      HBITMAP
Gradient    CSTRING('Gradient.bmp')


  CODE
  Bitmap = LoadImage (SYSTEM(PROP:AppInstance), Gradient, IMAGE_BITM
  ! We can use CreateSolidBrush to create a solid coloured brush or (
  !to create a brush with specific styles (like hatched) or CreatePat
  Brush  = CreatePatternBrush (Bitmap)

  GlobalResponse = ThisWindow.Run()                                ! Opens tl

  DeleteObject (Brush)
  DeleteObject (Bitmap)      ! when not needed anymore
  RETURN

!-----------------------------------------------------------------------------
DefineListboxStyle ROUTINE
!|
!| This routine create all the styles to be shared in this window
!| It's called after the window open
!|
!-----------------------------------------------------------------------------
Menu::Menubar ROUTINE                                              ! Code for
Menu::FileMenu ROUTINE                                             ! Code for
```

**Figure 3. The Clarion 7 IDE (pre-beta)**

## Starting with the Dictionary editor

Clarion is used for database-oriented business applications far more than for any other product category. And that makes perfect sense, since Clarion was designed, a quarter of a century ago, as a business application language.

Unless you're one of those rogue developers using Clarion for system utilities or some other sick purpose, you're going to begin your development process by either defining a set of files (tables) in the Dictionary Editor, or by importing a bunch of definitions from an existing database. Again, I'll have much more to say about this process in a future article.

## The AppGen and the templates

The Application Generator is the part of the IDE that most Clarion developers use most of the time. It's not strictly necessary to use the AppGen; some developers only write code by hand, but as Clarion's code generation has become more powerful, fewer and fewer Clarion developers have found the need to write everything themselves. It's a shame, really; there's much to be learned (and appreciated) by writing even a tiny business application one line of source at a time. But that's progress, and who has that kind of time anyway?

So let's assume that the AppGen is going to write most of your code for you. How does it know how to do that?

First of all, you normally associate every application (.APP file) you create with a dictionary (.DCT file). Again, that assumes you've more or less sorted out your database design before you begin writing any actual code; you can try it the other way but I won't promise good results.

When you create a dictionary, what you're really giving the AppGen is metadata – that is, data about your data. You're describing not just the individual fields (columns) and files (tables) but also field validation rules and the relationships between various files. The rule of thumb is that effort expended in getting your dictionary right *always* pays dividends when it comes time to build the application.

While it's possible to create an APP without a DCT, the AppGen can only automatically create applications for you if it has a DCT to work with. The AppGen uses a set of special templates, called wizards, to build such an app. A wizarded app is unlikely to meet all your design criteria, but it can let you view, update and report on all the data in your database, according to the validation rules and relationships you've defined in your data dictionary. You can also tune the wizards to your special requirements – some developers go through numerous cycles of wizarding up an app to get the wizard settings just right before creating the app they actually use. As with getting the dictionary right, tuning the wizards can have a big payoff.

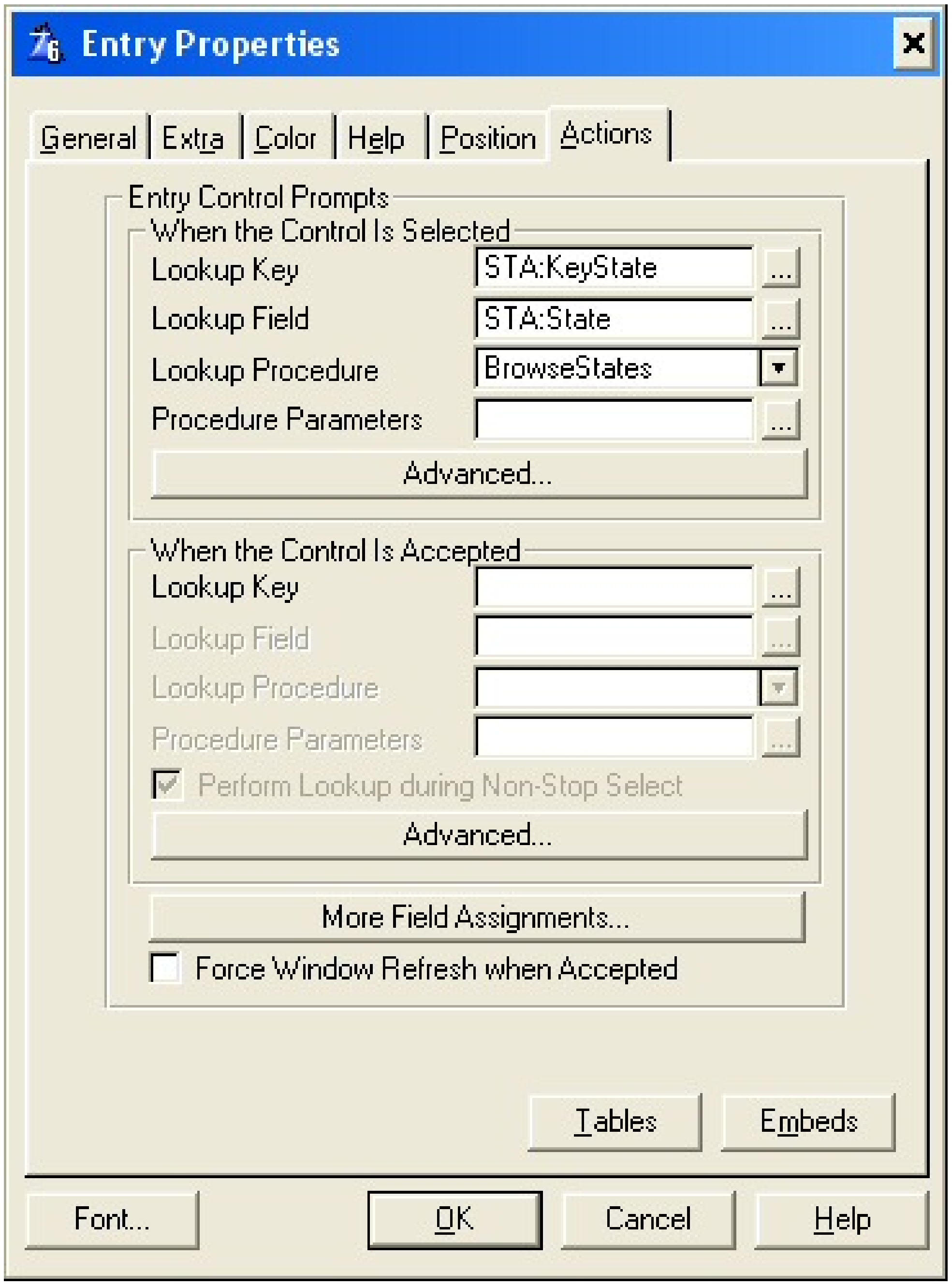
The alternative to the application wizard is to manually add procedures one at a time, but even here you have the option of using procedure wizards to speed up the process. If you want total control, you can always create simple windows and populate them with all the desired controls. You can choose your comfort zone in AppGen, from highly automated app construction, to nearly manual app construction.

Once you have some sort of application in hand you're ready to enhance it and add needed functionality. For the most part, you’ll do this via template prompts, or by embedding source code.

## Enhancing the application via template prompts

I’ll assume that you already have an application open in front of you, either yours, or one of the Clarion example apps. What follows is one small example of enhancing an application via the templates.

Click on any procedure with (Form) after its name. On the right hand pane you should see a tree list with top level items including Tables, Data, Window, Embeds and Extension. The traditional way to get to a window control's properties is to double-click on the procedure properties, then click on the Window button, then right-click on the desired control. In Clarion 6 you have a couple of shortcuts available, using the right hand pane, as shown in Figure 1. You can double-click on the Window line to get the window formatter and from there to the control, or double-click directly on the control's line in the right-hand pane. Do so now for any entry field and you'll get a dialog window something like Figure 4. Select the Actions tab.


Entry Properties

General | Extra | Color | Help | Position | Actions

Entry Control Prompts

When the Control Is Selected

| | |
|---|---|
| Lookup Key | STA:KeyState |
| Lookup Field | STA:State |
| Lookup Procedure | BrowseStates |
| Procedure Parameters | |

When the Control Is Accepted

| | |
|---|---|
| Lookup Key | |
| Lookup Field | |
| Lookup Procedure | |
| Procedure Parameters | |

[✓] Perform Lookup during Non-Stop Select

[ ] Force Window Refresh when Accepted

**Figure 4. Entry field Action settings**

The dialog in Figure 4 allows you to do field validations by looking up the contents of the entry field in another file. You could write the code yourself, for this and every other field that needs it. Hey, that's what cut and paste is for, right? On the other hand, since most of the code is the same for every lookup, and only the variable/key/file names change, why not just plug those in and generate the code automatically? And that, of course, is just what the template system is for. You don't need to understand exactly what this template does just yet – the point here is to grasp the basic mechanism by which the AppGen works.

Here's a fragment of the template code that displays the prompts:

```
#PROMPT('Lookup Key',KEY),%PreLookupKey
#ENABLE(%PreLookupKey),CLEAR
  #PROMPT('Lookup Field',COMPONENT(%PreLookupKey)),
    %PreLookupField,REQ
  #PROMPT('Lookup Procedure',PROCEDURE),
    %PreLookupProcedure,REQ,
    PROP(PROP:DropWidth,140)
  #PROMPT('Procedure Parameters',EXPR),
    %PreLookupProcedureParameters
#ENDENABLE
```

And here's a fragment of the template code that generates the actual source code:

```
%PreLookupField = %ControlUse
IF Access:%File.TryFetch(%Key)
  #IF(%PreLookupProcedureParameters)
  IF SELF.Run(%(INLIST(%PreLookupProcedure
    &'('&%PreLookupProcedureParameters&')'
    ,%ProcsCalled)),SelectRecord) = RequestCompleted
  #ELSE
  IF SELF.Run(%(INLIST(%PreLookupProcedure,
    %ProcsCalled)),SelectRecord) = RequestCompleted
  #ENDIF
    #FIND(%Field,%ControlUse)
    #FOR(%Relation),WHERE(%RelationKey = %PreLookupKey)
      #IF(%FileRelationType = 'MANY:1')
        #FOR(%FileKeyField),WHERE(%FileKeyFieldLink)
          #IF(%FileKeyFieldLink = %PreLookupField)
            #BREAK
          #ENDIF
    %FileKeyField = %FileKeyFieldLink
        #ENDFOR
      #ENDIF
    #ENDFOR
    %ControlUse = %PreLookupField
    #INSERT(%MoreAssign)
    #SUSPEND
#?ELSE
    #INSERT(%ClearAssign)
    #RESUME
  END
END
```

And finally, here's some code that this template code might actually generate, assuming you filled out the prompts:

```
STA:State = NAM:State
IF Access:States.TryFetch(STA:KeyState)
  IF SELF.Run(1,SelectRecord) = RequestCompleted
    NAM:State = STA:State
  END
END
```

Do you need to know, or understand, the template code I showed in order to be a productive Clarion programmer? Absolutely not. The reason I showed you that code was to illustrate the degree to which the AppGen is configurable. Most (though not all) of the prompts you'll encounter while developing your apps are there because somewhere a template says "show such and such a prompt." And the output of the AppGen is *completely* under the control of the templates, which is why it's possible to create templates that generate code for C++, Java, C#, or any other language. For instance, Fenix is a template set for generating ASP.NET applications using C# or VB.NET.

And templates are just text files so you can create your own. Yes, there's another language to learn (most template symbols are prefixed by the # character), and templates are actually a mix of Clarion and template language statements, which typically makes them a bear to read, especially when you're dealing with two different sets of indents. But the template system really is the heart and soul of Clarion. You can do worse than lean how to write templates.

## Enhancing the application via embed code

The second major way to add new behavior to your program is to embed snippets of Clarion code in your procedures. You do this by using one of several embed editors (which I'll cover in more detail in a coming article on AppGen). Back in the DOS days, Designer (the ancestor of AppGen) offered a very limited number of places where you could insert your own code. This was a cause of much gnashing of teeth, and not a few workarounds. These days, however, there are embed points aplenty.

In general, the idea behind Clarion development is to use templates for all repetitious code, and embed points for one-off code. Few developers, of course, have never done a cut and paste job on some code, and there are other ways of accomplishing code reuse, such as procedure libraries and object orientation, both of which Clarion supports. But from the perspective of an AppGen user, it's almost all about template prompts and embed code (and, of course, the data dictionary).

Capesoft's Bruce Johnson tells me that the number one question he hears from new Clarion developers is: "Where is the documentation that tells me what each embed point is for?" And as Bruce points out, the answer is that there is no exhaustive list, simply because there are way more embed points than you will ever need, and many if not most are there simply because they can be there, not because they serve a useful purpose. The most-used embed points include those for control event handling (when a control is selected or accepted, or when its contents change), procedure routines, and procedure startup and shutdown. In a later article in this series I'll discuss embed points in more detail. For now you may want to read Tom Giles' article ABC Embeds are Easy.

## Next time

So far I've provided a very brief summary of the major pieces of the IDE, and as I said earlier I'll be dealing with each of these in more detail in upcoming articles. There are still a few important concepts to cover, which I'll get to next time. These include the browse/form paradigm, procedural vs. object-oriented code, and the database drivers.

## Suggested Reading

- Getting Started – Using Clarion (topic)
- Templates, general info (topic)
- Templates, writing (topic)
- Tom Ruby's excellent series on normalizing databases
- Tom Giles' ABC Embeds are Easy

---

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

# Solving Sudoku Puzzles With Clarion, Part 2

## by Jon Waterhouse

Published 2006-09-27

In Part One I explained the code for solving Sudoku puzzles with logical solutions. But there are some tricky Sudoku puzzles which cannot be solved directly with logic. Well before you have found all the values in the grid, the `run_logic` procedure tells you it has found nothing this round. When you hit that point the only way to solve the puzzle is to try inserting one of the values that is still valid into the grid and see whether you can reach a solution. You may have to make several guesses. In the puzzle shown below, three guessed values will get you to the solution.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | 3 | | |
| | | | | 2 | 8 | | 6 | 9 |
| | | | 9 | | | | 1 | |
| | | 9 | 2 | | | | 7 | 8 |
| | | | 7 | | 1 | | | |
| 8 | 6 | | | | 5 | 4 | | |
| | 8 | | | | 2 | | | |
| 1 | 2 | | 5 | 6 | | | | |
| | | 4 | | | | | | |

Logic will only get you as far as the grid below. Of course there are also many other constraints found that just haven't allowed you to fix the value of individual cells yet. The cells in bold are those that the program will guess in order to complete the puzzle.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **9** | | | | | 3 | | |
| | 3 | | | 2 | 8 | 7 | 6 | 9 |
| | | | 9 | | | | 1 | |
| 3 | 1 | 9 | 2 | 4 | 6 | 5 | 7 | 8 |
| | | | 7 | 8 | 1 | | | |
| 8 | 6 | 7 | 3 | 9 | 5 | 4 | 2 | 1 |
| | 8 | | | | 2 | | | |
| 1 | 2 | 3 | 5 | 6 | | | | |
| | **5** | 4 | | | | | **9** | |

When you are faced with having to start guessing, rather than using logic, a number of things change.

First, in the logic zone, unless people kept on giving you grids that contained starting values that could not lead to a solution, you don't really have to worry about the possibility that constraints were in conflict. But when you start guessing values, contradictions will happen, and it is important that the program catches and signals contradictions as soon as possible.

Second, you need a way to get back to the situation you were in before you made a bad guess.

Third, you need a system for guessing that, at a minimum, is guaranteed not to cycle and, although this is not of great importance for the 9x9 puzzle - that also leads to a solution quickly.

## Flagging the impossible

In Part One I indicated some of the code paths that don't lead to a solution.. Here is a list of all the places where a conflict can occur:

1. You find you have tried the last potential value of the first cell you guessed and still have no solution (in `back_up`)
2. Your "total" values in `check_grid` show that you have two cells with the same value with in a triplet group.
3. You try to add more than 6 values to `out_trips` for any triplet (`add_to_outs`)
4. You try to add more than 3 values to `in_trips` for any triple (`add_item`)
5. You try to add a value to a triplet that is in `out_trips (rejected_value)`
6. You try to add a value to a triplet that is already in `in_trips (rejected_value`)
7. You try to add a value to a triplet in an impossible place (either there is another value already there, or the value you are adding is already in another position) `(fixpos_accept)`
8. A value that you are adding in a triplet appears in a cross triplet that already has a value that is in your triplet, like in the example. Say you are adding the value 3 in row triplet. This cannot be, because it would have to share the same position as the value 1. `(check_for_fp)`

| | | | | |
|---|---|---|---|---|
| 1 | 2 | 3 | | 1 |
| | | | | 7 |
| | | | | 3 |

When these errors are found the error flag needs to be passed back up the line so that the procedure handling the recursion knows it has to toss the value it is working on and try another.

## Getting back to where you were

There are really two ways of going back to a previous point. One is to keep track of the "rules" added as a result of each guess. When I have to abandon a guess I remove the rules that were contingent on that guess. The second method is to store a copy of what the constraints looked like before each guess, and restore the copy if the guess is revoked. I opted for the first approach.

With the exception of the fixed position constrraints, constraints are always added to the end of the existing list of rules. For example, I add a third value to `in_trips` as a result of a particular guess. If I have to revoke the guess, all I need to do is note that I now have only two useful values in `-in_trips`. Although fixed positions *could* have been handled the same way, they weren't, so I actually have to blank the relevant values in the fixpos array. Way back, I mentioned that I would explain the importance of the second parameter to `find_in_v` later. Later is now. Since I am leaving obsolete junk at the end of `in_trips` and `out_trips` I need to make sure that procedures like `find_in_v` look only at the valid items.

Keeping track of what needs to be removed is very simple using a queue. Each queue element (in `items_added)` contains the guess number (`level`), the type of addition (in,out or fixpos), the index of the triplet (r,c,n,t) and, for fixpos additions, the position in the array. Once I have blanked the fixpos or decremented the counter for in or out I can delete the queue entry.

## Guessing strategy

There are probably two main ways to manage the guessing strategy. One is to look at all the values you may have to guess right when you start guessing, then work your way methodically through the static list. Obviously, as you get further down the guessing tree there will be values in your list that are no longer valid given your previous guesses. You don't waste much time on them, but if you have a large problem (16x16 or higher) it might add up.

The second method is to choose which cell to guess next depending on the situation prevailing after the previous guess. This is what I opted to do. I'll explain why.

My theory is that if you are on a bad path, you want to be on a short bad path, so that you can get back off it as fast as possible and onto a better, hopefully the right, path. I figure you increase your chances of finding conflicts on a bad path sooner if you:

1. pick the cell with the fewest choices left at each step, and

2. choose cells that are close (same box, row and/or column) to the previous cell you chose.

For example, say the first cell for which you guess is (1,1), which has two possible values. You take one value as your guess and chase down any new rules that arise out of this guess. Then you look at which cells now have the fewest potential values, and if there is a tie, you choose one that is close to (1,1). Say it is (1,3). You follow along that route, and it turns out to be a bad one. You end up back at (1,1) and pick the second value that was possible there. You chase down new rules based on the new value in (1,1), but this time the best pick (because it has fewest possible values) is cell (8,1). You follow that path, and somewhere down there (or in the path from any other allowed value in (8,1)) you should find the solution.

Although I have called the guessing manager `Manage_recursion` it is actually iterative. The procedure (minus some unimportant details) looks like this:

```
manage_recursion procedure
rv2 short
k short
code
  if prepare_recursion_level(1) = -1
    return
  end
  loop
    loop_counter += 1
    rv2=addvalue(r_q.row,r_q.col,r_q.ps.possibles[r_q.item])
```

```
        if rv2 >= 0
          rv2=run_logic()
          if rv2 >= 0
            rv2 = fixed_pos_corr_all()
          end
        end
        if rv2 < 0
          ! Found a problem, need to try another value
          if back_up() < 0
            message('Recursion: no solution found')
            break
          end
        else
              !All logic has been run with tested value, run_logic has returned 0
          !Time to test an additional value
          if prepare_recursion_level(r_q.level + 1  ) = 1
              break
          .
        end
      end  !loop
      if rq:level > 0
        message('Recursion: solution found!')
      end
      rv=fill_display_grid()
```

Firstly, remember that run_logic finishes with a call to `fill_intersect`, which is a procedure that uses the intersection of the row and column triplets to figure out which values have been found and how many possible values there are for cells that have not yet been found. `Prepare_recursion_level` uses that information to find the cell with the smallest number of allowed values and, if there is a tie, chooses the cell closest to the last cell guessed. The code then enters and endless loop:

- Add the guess
- If that goes OK, call `run_logic`
- Call fixed_pos_corr_all, which implements Rule 3 for every triplet – add_item and add_to_outs seem to miss some opportunities to propagate constraints as they go
- If there's a problem call `back_up` to, unsurprisingly, back up. What back_up does first is get rid of any constraints added as a result of the last bad guess. Then if there are still values left to try in the current cell, it tries one of those, otherwise this cell is abandoned and the code backs up to the previous cell that was guessed. As `back_up` navigates to the next value to try it calls `rem_q_items` to remove rules that were dependent on guesses that are being abandoned.
- If there were no problems with the value just added, but the solution still hasn't been found, then call `Prepare_recursion_level` to set up the next cell to guess. If all cells have a found their value then the code is done, otherwise
- Go back to the top of the loop again.

The only other noteworthy thing in the guessing portion is that it is important not to start guessing the same cell twice. The 9x9 `intersect` group contains a marker called `off_limits`. As a cell is guessed it is marked as off_limits (in `Prepare_recursion_level`). If I back up past that cell (in `back_up)`, the marker is removed.

That's about it. The construction of the grid is sort of neat, too. I didn't like the thought of having to manually add a whole bunch of little boxes for people to type in their starting values. 81 for the 9x9, 256 for the 16x16, not to mention the size of the screen definition, so I wrote a little loop to create the entry fields dynamically and then draw the lines between them in the appropriate places.

## Last words

I'm not really sure why anyone would want a Sudoku solver. However, making a solver is an interesting exercise for some sorts of people (like me), not that dissimilar from actually solving a Sudoku. And it's obviously a deep-seated problem. I also felt compelled to write a program to solve Mastermind problems (that game with the coloured pegs where you had to guess the code) many years ago, in C.

The Sudoku program contains a couple of 9x9 grids and a 16x16 grid in the grid_setup procedure. Two are currently "omitted". You press the "Grid Setup" button to fill in the grid from the stored values, or you can enter your own puzzle (from the newspaper, or wherever), and press the "Accept Grid" button. A feature is that when you press either of the Grid buttons, in addition to the values being added into their positions, the corresponding entries are made to the `outs`. On occasions, the constraints added in this process are enough to fix new values in the grid, before you have even really started the solver. They'll appear when you press the button.

To set up the program to solve 16x16 grids, just change the SZ EQUATE to 4 at the top of the program.

Two things that my solver currently doesn't do, but which wouldn't require much work.

1. If you wanted to generate Sudoku puzzles you would want to be able to save the grids you were working on. It would probably also be useful to ask the program: "Can I take away any of the initial grid values and still get to the solution without having to guess?" Not hard to do.
2. Possibly related to the desire to generate puzzles, or for other reasons, you might want to modify the program to find not just the first solution for puzzles that require guessing, but to check for possible multiple solutions. That's not too hard to do either. All you would have to do is, having found the first solution, get the program to backtrack to your last guess, pick the next value for that cell, restart the solver and look for more solutions.

Download the source

---

Jon Waterhouse has been using Clarion since the 2.1 days. His main work is as an economist, and he finds that Clarion is well-suited for applications which impose order on various sets of data. His projects include questionnaire data entry programs, classification software (assigning projects to groups), plus some more interesting scheduling applications. Jon has also used Clarion to link text information together, and is currently developing a program that will store linked snippets of WordPerfect documents and print custom documents composed of several of these snippets. He is currently working for the Newfoundland Government on a project to measure the performance of government employment programs.

## Reader Comments

Add a comment

Clarion Magazine

# Solving Sudoku Puzzles With Clarion, Part 1

**by Jon Waterhouse**

Published 2006-09-21

Sudoku are cute little puzzles that first became popular in Japan. The rules are very simple: you have to fill in a 9x9 grid with a set of numbers already filled in, such that every column, row and 3x3 box contains the digits 1 through 9.

To solve the puzzles you mostly use logic; some of the harder puzzles cannot be solved directly from the information given – at some point you have to guess at a couple of numbers, and if they don't lead you to a solution you have to backtrack and guess again.

If you've done the puzzles by hand you probably have an idea of how to go about finding a solution: on one side if you know that a particular number cannot go some place, you know that it must go in another. Approaching it from the other side, since you know that all of the numbers must exist in a particular row/column/box and a particular number is not yet there, then that number must go in one of the unfilled spots.

Also, when you do it by hand, you mostly care about what values are valid for each cell; in my program the main unit I care about is the triplet; a group of three cells in a row or column within a box. I'll get to why I did that in a little while.

My computer program basically follows a deductive reasoning approach, but that is not the only possible approach. You could also use a brute force approach. In that approach you make oodles of guesses at what the solution might be until you find one that fits. It can be done, but think about the cost. It's like the apocryphal story of the guy who invented chess asking to be paid $2^n$ grains of rice for each of the 64 squares on the board. If you have 81 squares, with 9 possibilities for each square, with about 30 given to you at the beginning, how many combinations do you have to test? My estimate would be that if you have about 5 values per box to find, the number of combinations in that box is 5!, or 120. You have 9 boxes, each of which can have 120 values, which is about 5*10^18. If you can apply logic you probably get to the solution much quicker.

Interestingly, Prolog, which you might think of as a good language for logic, allows you to write a very small program to solve a sudoku, but the approach that it takes is basically the brute force method. What Prolog will not natively do is what is called "constraint propagation". Without this technique, Prolog guesses at a number for a cell and asks: is this number consistent with the three rules and the fixed values I got at the beginning in combination with previous guesses I made? However, as I shall now demonstrate, that approach has you looking at possibilities that logic would already have ruled out.

My constraint propagation approach relies on four rules.

**Rule 1: If a value is in one triplet in a group, then it cannot be in the other triplets in the group**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | | | | | | |
| ▨ | ▨ | ▨ | | | | | | |
| 2 | | | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

**Rule 2: If a value does not exist in two other triplets in a group, it must be in the remaining triplet.**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ▨ | ▨ | ▨ | ▨ | ▨ | ▨ | | | |
| ▨ | ▨ | ▨ | | 2 | | ▨ | ▨ | ▨ |
| 2 | | | ▨ | ▨ | ▨ | ▨ | ▨ | ▨ |

Here you can see that the two given values of 2 rule out the possibility that the value 2 exists in all of the shaded triplets (by Rule1). Based either on the fact that 2 is outlawed from the other triplets in box 3, or that it is outlawed from the other triplets in row 1, Rule 2 says that the value 2 must be in the first row of Box 3.

**Rule 3: A value can be outlawed from a box column by a combination of restrictions on the box rows that cross that column and fixed position constraints within the box column.**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | | | 5 | | |
| | | | | 2 | | | | |
| 2 | | | | | | | | |

By rule 2 above, I determined that there has to be a 2 in the first row of the third box. However, for the first column in that box, the position in the first row is already occupied, and the value 2 cannot go in either row two or row three, based on Rule 1, so the value 2 must be added to `out` list of the first column of the third box.

At this point you may be thinking: what's all this business with triplets? If you were just dealing with cells you'd simply say that you couldn't put a 2 in the six cells from 2,7 to 3,9. The deal with triplets, though, is that in addition to being able to express the *exclusion* rules (value 2 cannot be in cell 2,7), you can also express the *inclusion* rules (value 2 *must* be in row triplet 1 in box 3). And that is an important constraint to be able to store.

Also, just think for a second about how you would code stuff if the primary unit for rules was the cell. For each value in the initial grid you would record 20 out constraints (8 in the row, 8 in the column and 4 additional within the box), not the 8 for triplets. And you would also still need to collect information at the triplet level. Consider the situation below:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | 1 | 2 | 3 | | | |
| | 7 | | 4 | 5 | 6 | | | |
| | | | | | | | | |
| | | | | | | | 7 | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | 7 |
| | | | | | | | | |
| | | | | | | | | |

In this diagram the third row in the second box must contain the values 7,8 and 9 (Rule 2). They could be in any column, but they all have to be in that row. Combine that with Rules 1 and 2 on the 7s in columns 8 and 9, and box 1,7 must be a 7. You cannot really come up with this conclusion on a cell basis alone. Sure, you know that each of those three cells in the third row of the second box will ultimately be a 7,8 or 9, but that collection of three cell facts does not rule out the possibility that 7 is also a potential value for another row in the box.

***Rule 4: If you have three in values for a triplet, then all the other values must be outs, and conversely, if you have 6 out values you know the in values***.

So now let's get down to some details of the Clarion program.

Start with the triplets. Each row has three and each column has three, so there are 54 of them. I decided the easiest way to number them was by box row, box column, triplet within the box, and whether the triplet was a row or a column triplet. The row triplet that takes in cells (5,4) (5,5) and (5,6) is the second row of boxes, second column of boxes, second triplet down and is a row, so it is triplet[2,2,2,1]. The column triplet with cells (7,3),(8,3) and 9(3) is row 3, column 1, item 3 and a column, so it is triplet[3,1,3,2].

When you number them that way, then the three triplets that make up a box are the same box row and box column, and you can loop through the items. The three triplets that make up a column are the same box column, same item, so you loop through the box rows.

Each triplet needs to be able to store three `in` values and six `out` values. I have these in groups labeled in_trips and out_trips. You also (in order to implement Rule 3), need to know if any of the `in` values have a fixed position. The `in_trips` group therefore also has an array to store the position of values that are fixed within the triplet.

I'll admit that when I started writing the code I thought that you could probably work out solutions to the column triplets and row triplets separately, then generate the final grid by finding the intersections, and not worry about fixed values trying to solve the rows and columns separately. It turned out to be a little more complicated than that, so the fixpos array is a bit of an afterthought, but it works fine.

So, how do things get in to the `in` and `out` values for a triplet? Let's say that the initial grid puts a 6 in position (1,1). I add a 6 to the in_trips[1,1,1,1] (the row triplet) and in_trips[1,1,1,2] (the column triplet). At the same time I add a 6 to the following out_trips: [1,2,1,1],[1,3,1,1] (they are in the same row), [2,1,1,2] and [3,1,1,2] (they are in the same column), [1,1,2,1] and [1,1,3,1] (they are the other row triplets in the same box) and [1,1,2,2] and [1,1,3,2] (the column triplets in the same box). That's just the implementation of Rule 1.

When we have carried out that process for all of the given values, hopefully there will be a set of triplets where two of the triplets will have `outs` for a particular value, Then the values must be *in* the remaining triplet. The way  I have programmed this is possibly not the most efficient way, but it is pretty easy to understand. Say I am looking at a row, and the first triplet in the row has an `in` value of 1 and five out values (3,4,5,6 and 7). The second triplet has an in value of 5 and out values of (1,3,6 and 7) and the third triplet has an in value of 7 and out values of ( 1,5).

I transfer these into an array that has three columns (one for each triplet) and nine rows (one for each possible value). If the triplet has an `in` value of that number, I place a 4 in that position. If it has an `out` I place a 1. The following table shows this data.

| **Value** | **Triplet 1** | **Triplet 2** | **Triplet3** | **Total** |
|---|---|---|---|---|
| 1 | 4 | 1 | 1 | 6 |
| 2 | | | | |
| 3 | 1 | 1 | | 2 |
| 4 | 1 | | | 1 |
| 5 | 1 | 4 | 1 | 6 |
| 6 | 1 | 1 | | 2 |
| 7 | 1 | 1 | 4 | 6 |
| 8 | | | | |
| 9 | | | | |

Look at the total column. I get a 6 where I have an `in` value. This is because I did my homework when I added the in value; I made sure that there was a corresponding out value in all the other triplets in the group, so 6 is the total I want to see. I don't need to do anything further with those values. For values 3 and 6 I have totals of two. This means the first two triplets have outlawed that value, so the third triplet has to contain these values (Rule 2). So I get to add 3 and 6 to the `in_trips` of the third triplet, and to the corresponding `out_trips` of the other triplets in the groups that third triplet belongs to. The total of 1 (for value 4) I ignore: I don't have enough information to do anything.

Are any other values possible? No – at least not that are consistent with good housekeeping and a solution to the problem. Taking  size (SZ) to be  3, the meaning of the total values are:

- anything less than SZ-1 you don't have enough information to do anything.
- SZ-1 means you have found an `in` value.
- SZ is an impossibility (a value cannot exist in any of the triplets in the group)
- SZ+1 through 2*SZ-1 are errors in propagating constraints you should never see
- 2*SZ is a completed group
- Anything more than 2*SZ is an impossibility (a value exists twice in a group).

Why the SZ business? Well, I wrote the program with the intention that it would also be able to solve what are sometimes called "Super sudokus" which have a 16x16 grid. For those puzzles SZ=4. In the code you will see a fair number of references to SZ and SZ2, which are 3 and 9 respectively for the 9x9 puzzle. The process above is carried out by the `fill_checkgrid` procedure, which fills the array, and `check_grid`, which acts on the results.

Getting back to propagating constraints: that is what I am doing when I add values to the `in_trips` and `out_trips`. The `in_trip` constraints are that triplet `y` *must* contain the value `n`. The `out_trip` constraints are that triplet `y` *cannot* contain the value `n`.

With that background, let's look at a little bit of the code, starting with the `add_to_outs` procedure that, you guessed it, adds a value to `out_trips`.

```
add_to_outs procedure(r,c,n,t,val)
i  short
rv2 short
  code
  if find_in_v(out_trips[r,c,n,t].v,triplets[r,c,n,t].out,val)
    return 1
  end
  if triplets[r,c,n,t].out>= SZ2 - SZ  ! e.g. 9 possible minus 3 in values
    return -1                          ! Trying to add too many outs
  else
    triplets[r,c,n,t].out += 1         ! if not then add it
    out_trips[r,c,n,t].v[triplets[r,c,n,t].out] = val
    if transfer_out(r,c,n,t,val) < 0 then return -1.
    if vals_left(r,c,n,t) < 0  then return -1.
  end
  return 0
```

`find_in_v` is a little function that looks for the third parameter in the elements of the array passed in the first parameter up to the element number specified in the second parameter. If it found it, it would indicate an attempt to add an out value that is already out, so `add_to_outs` just returns. The importance of the second parameter in `find_in_v` will be explained later.

Next it checks for an impossibility that might happen on a bad guess towards a solution: more than 6 values have been eliminated for that triplet, so there are less than 3 values to populate the three spaces. Here a wrong guess has led us to this situation, and back-tracking will be necessary. The procedure shows this by passing back a negative return value. I've taken out a couple of the other lines of code that are only necessary when you have to guess. That part will be explained later on.

If the code passes the first two hurdles he procedure increments the number of values in the out array by 1 and adds the value into the next available position in the array.

With the new value added, `transfer_out` implements Rule 3. Adding the `out` to a row (or column) may mean that the out value also needs to be added  to one of the columns (or rows) that crosses this one in the same box.

Finally the call to `vals_left` deals with the desirable situation where there are 6 `out`values, which nails down the `in` values. `Vals_left` calls `add_item_internal` to add items to `in_trips` if they are not already there.

The next procedure to look at is transfer_out:

```
transfer_out procedure(r,c,n,t,val)
unfree_cols byte,dim(SZ)
exclude_from byte,dim(SZ)
num_excluded short
one_fixed byte
no_good_cols short
i short
j short
rv2 short
  code
  exclude_from[n]=1
  !Documentation based on the original t=1 (a row)
  !This part collects all the rows that outlaw this value
  loop i = 1 to sz
    if i = n then cycle.
    if find_in_v(out_trips[r,c,i,t].v,|
          triplets[r,c,i,t].out,val) then exclude_from[i]=1
        end
  end
  !For each column I now need to find out which
  !cells this value would be excluded from
  !First blank the array
  loop i = 1 to sz
    loop j = 1 to sz
      unfree_cols[j]=0
    end
    loop j = 1 to triplets[r,c,i,t%2+1].in
      if in_trips[r,c,i,t%2+1].fixpos[j] <> 0 |
              and in_trips[r,c,i,t%2+1].v[j] <> val
        !If fixed, and not the value I am examining
        unfree_cols[in_trips[r,c,i,t%2+1].fixpos[j]]=1
      end
    end
    no_good_cols=0
    loop j = 1 to sz
      if unfree_cols[j]=1 or exclude_from[j]=1
         !Cell can be eliminated for either reason
             no_good_cols += 1
          end
    end
    if no_good_cols = sz
      !We add to outs only if all cells in the
          !column are eliminated
      if add_to_outs(r,c,i,t%2+1,val)  < 0 then return -1.
    end
  end
```

```
        return 0
```

`Transfer_out` implements Rule 3. First note that `t%2+1` changes from a row (1) to a column (2) or vice versa. Say I've just added the value 7 to the `out` values for first row (1) in a particular box. This procedure is going to determine if it should also be added to the `out` values of any of the column triplets in the same box. The value is excluded from any position 1, which is the reason I am calling the procedure. Next the code looks to see if this value is outlawed by any of the other rows in the same box. If it is it documents that in the matching element in `exclude_from`.

Now, for each column the code looks to see if there are specific positions that the value cannot go. If so, these are documented in the `unfree_cols` array. Finally it loops through each position in the column. If *all* the positions are eliminated then the code calls `add_to_outs` to outlaw the value from the column. Some of the code here is a little trickier than it would have to be if I stored constraints at the cell level, but I hope I've convinced you already that the triplet is the level at which most constraints should be stored.

`Add_item`is the procedure that deals with adding a value to a triplet. It's probably the most complicated bit of code. Again, part of the complication is due to the fact that I am dealing with constraints at the triplet level. Let me explain why.

You have a triplet for which you know all three values, and you know which position one of those values is in. Say the values are 1,2 and 3, and you know that the 3 is in position 3. Your `in_trips` group for the triplet could look like this:

| | | | | |
|---|---|---|---|---|
| V | 1 | 3 | 2 | |
| Fixpos | | 3 | | This indicates that the value 3 is in position 3 |

Now, say that the guessing part of the program has proposed that position 1 in this triplet should have the value 3. Ignore for now that it should probably not be allowed to guess that; the third column triplet should have 3 as an `in` and the other two column triplets should have 3 as an `out`. If such a thing happened, I need to ensure that the proposal is rejected. Even though 3 is a valid value to go in the triplet, adding it in two positions in the same triplet is not allowed. This requires a little check to make sure that if the procedure is called to add a value to the triplet with a fixed position, that the value does not already exist in another fixed position in the triplet.

So assume that the procedure has been called for the triplet above with a value `val` of 1 and a fixed position `fp` of 2. The value 1 already exists in `v`, all I need to do is add a 2 to `fixpos[1]`. The checking and the addition of a `fixpos` value are handled by `add_fixpos`, which is the first item in the procedure. `add_fixpos` does a check for things that are not consistent. If an inconsistency is found a negative value is returned and this causes `add_item`to terminate.

If I am not simply fixing the fixpos for a value already in the triplet, I enter the more complicated part of the procedure. I have a bunch of checks for consistency:

1. That there is room for another value
2. That the value being added is not in `outs`
3. That the value being added is not already in the `ins`
4. That the `fixpos` (if any) specified for the new value does not cause internal conflict in the

triplet or violate existing column rules (for a row)

If there is any violation, the code returns an error, otherwise it adds the value. If the new value completes the triplet it calls `triplet_full` to implement Rule 4 (make sure all the non-`in` values are in the `outs`, and to fix the last position if it knows two of the three positions in the triplet).

To complete the procedure there is a call to `fixed_pos_corr` which is another implementation of Rule 3 and `post_add`which implements Rule 1 (putting the value in the `outs` for all other triplets in the group).

```
add_item procedure(r,c,n,t,val,fp)
rv3 short
j  short
z short
temp short
nextpos short
  code
  ! First assume I am just adding a fixpos
  ! to an existing value in a triplet
  rv3= add_fixpos(r,c,n,t,val,fp)
  if rv3 < 0 then
    add_debug(1,r,c,n,t,val,fp)
    ! Failed - abort
    return rv3
  elsif rv3 = 0
    ! Was a real add - continue
    if triplets[r,c,n,t].in>= sz then
      add_debug(2,r,c,n,t,val,fp)
         ! No room left in triplet - abort
      return -9
    end
    if rejected_value(r,c,n,t,val) then
      add_debug(3,r,c,n,t,val,fp)
      ! Value is impossible for triplet - abort
      return -8
    end
    ! Prepare to add value, but don't add till later
    nextpos =  triplets[r,c,n,t].in + 1
    if fp> 0
      ! If adding an item in a fixed position
          ! Check to make sure it can go there
      if fix_pos_accept(r,c,n,t,nextpos,val,fp)=1
        in_trips[r,c,n,t].fixpos[nextpos]=fp
        add_q_item('X',r,c,n,t,nextpos)
      else
        add_debug(4,r,c,n,t,val,fp)
            ! if not, then abort
            ! Adding item caused a fixpos conflict
        return -6
      end
    else
```

```
            ! If adding to a column, then check rows and vice versa
            temp= check_for_fp(r,c,n,t%2+1,val)
            if temp> 0
                  ! A matching value was found in a cross-triplet
                  ! So add the fixpos value
              in_trips[r,c,n,t].fixpos[nextpos]=temp
                  ! For recursion
              add_q_item('X',r,c,n,t,nextpos)
                  ! And try to add the fixpos value to the cross-triplet too
              loop j = 1 to triplets[r,c,temp,t%2+1].in
                if in_trips[r,c,temp,t%2+1].v[j]=val
                    ! Check to make sure this is legal
                  if fix_pos_accept(r,c,temp,t%2+1,j,val,n)=1
                    in_trips[r,c,temp,t%2+1].fixpos[j]=n
                    add_q_item('X',r,c,temp,t%2+1,j)
                  else
                    return -5
                  end
                end
              end
            elsif temp < 0
                  ! Adding item caused a fixpos conflict
              return -7
            end
          end
          triplets[r,c,n,t].in =nextpos
          ! Value addition now after fixpos check
          in_trips[r,c,n,t].v[nextpos]=val
          ! For recursion
          add_q_item('I',r,c,n,t,nextpos)
        end
        ! Should only ever be equal - cascade constraints
        if nextpos>= SZ then rv1=triplet_full(r,c,n,t).
        ! based on full triplet and new fixpos values
        if fixed_pos_corr(r,c,n,t) < 0 then return -1.
        ! And cascade some more constraints
        if post_add(r,c,n,t,val) < 0
          return -1
        end
        return 1
```

As I look through this procedure I see things I suspect are not actually necessary. The direct call to `fix_pos_accept` looks unnecessary, but then I check and find that, in addition to making sure that the same value is not in some other position (which really should be impossible at this point),  it also makes sure you are not trying to put this value in a position that is used by something else.I think about whether the thing I told you to ignore earlier – that the program might try to add a new value 3 in a column other than 3 in this example – could actually happen. I note that when a `fixpos` is set for a row triplet, that nothing currently adds a value to the `in` of the affected column, so maybe it is possible.

Some people write beautiful code based on clear, beautiful thoughts. I generally manage to think myself about 85% of the way to the full solution and then tinker to get myself the other 15%. So when I found that the program didn't get me all the way to the place I could get to by doing the sudoku myself, I added an extra check or an extra run through some procedure until it did. I've tried it on quite a few grids now, and it hasn't failed, so…

The final procedure that is interesting to look at is `run_logic`, which calls itself recursively. The first part of the procedure checks for items that can be added to triplets by looking at each possible triplet group – 9 boxes both horizontally and vertically, 9 rows and 9 columns. If anything is added in this process it calls itself for another run-through. When it can find no more values to add it calls `fill_intersect` which deals with a human-readable version of the grid, and also works out all the valid values left for cells that have not yet been calculated. That process may find cells where only one value is now possible. If so, `run_logic` calls itself to do another round of checking. Ultimately `run_logic` either indicates that it has nothing more it can add (returns 0) or indicates that it found an inconsistency while performing its duty (returns -1).

The `run_logic` procedure just had a little face-lift. While thinking about Mike Hanson's brute force approach to a solution I realized that for both his approach and mine, if nothing in the neighborhood has changed there's no point in looking out of the window. When adding an `in`, `add_item` adjusts the neighbourhood by adding in the required `outs`. Nothing more to do there, but it is the repercussions from those outs that can lead to new discoveries. If no out values in the triplet group have changed, then it is not worth calling the `check_grid` procedure for that group. So I added a call from `add_to_outs` to a procedure called `friendly_reminder`. `Friendly_reminder` adds a queue item to tell the two groups (row and box rows, or column and box columns) the triplet belongs to that they the other members of the group need to look out the window and check for changes. Having checked, the queue item can be deleted. Of course, when you start the solver all the groups need to check themselves, so when you press the "Run Logic" button all groups are added to the queue.

```
run_logic procedure
FOUND_THIS_ROUND short
i                short
rv2  short
current_recs short
  code
  found_this_round=0
  current_recs = records(groups_to_check)
  loop i = 1 to current_recs
    get(groups_to_check,i)
    CASE grps:grp_type
    OF 'BR'
     rv2 = check_boxrow(grps:ref)
     OF 'BC'
      rv2=check_boxcol(grps:ref)
    of 'C'
      rv2=check_col(grps:ref)
    of 'R'
      rv2=check_row(grps:ref)
    end
    if rv2>=0
```

```
        found_this_round += rv2
      else
        break
      end
    end
    !Delete all of the queue records
    ! I have dealt with (may be some new ones
    !have been added during this process)
    loop i = 1 to current_recs
      get(groups_to_check,1)
      delete(groups_to_check)
    end
    if rv2 < 0 then return -1.
    if found_this_round = 0
      rv2=fill_intersect()
      if rv2=0
        return 0
      elsif rv2 < 0
        return rv2
      else
       rv2=run_logic()
     end
    else
      !Called recursively until either a 0
      !(no more changes to implement)
      !Or a negative (impossible situation) is found
      rv2=run_logic()
    end
```

At this point you should have a fairly good idea of how the program goes about solving a sudoku which can be solved by logic alone. The values given in the initial grid yield constraints that tell us some values cannot exist in other triplets. Some of those combinations of constraints allow us to define new `in`constraints, which generate further `out` constraints and so on.

In Part Two I'll take a look at sudoku puzzles which cannot be solved directly with logic.

Download the source

---

Jon Waterhouse has been using Clarion since the 2.1 days. His main work is as an economist, and he finds that Clarion is well-suited for applications which impose order on various sets of data. His projects include questionnaire data entry programs, classification software (assigning projects to groups), plus some more interesting scheduling applications. Jon has also used Clarion to link text information together, and is currently developing a program that will store linked snippets of WordPerfect documents and print custom documents composed of several of these snippets. He is currently working for the Newfoundland Government on a project to measure the performance of government employment programs.

## Reader Comments

Add a comment

Clarion Magazine

# Roll Your Own Web Search Engine

by James Albrecht

Published 2006-09-14

One of the shortcomings of internet search engines has been the inability to restrict searches to a predetermined set of websites. Suppose that you are looking for a car and would like to restrict the search to a handful of dealerships that you know and trust. If the dealerships are listed with the major auto sites, you are usually able to drill down by proximity to their postal code, but even then, it will be difficult to grab only the information you desire. If you're like me, you've probably bookmarked various sites that you check manually. This means loading the page into the browser, right-clicking, and searching for a keyword. Or, worse yet, you may end up scrolling through pages of irrelevant inventory.

There has to be a better way. And there is.

One way would be a SOAP interface, though this is not always available. Another would be to use www.rollyo.com or the new Microsoft search engine. Both promise this type of capability, though I have yet to test them out.

A third, and better solution is to write an app in Clarion. This will allow you to have complete control over the way the search code behaves. For example, you may want this app to fire automatically during the night so the results are waiting for you in the morning; you may want to store the retrieved information in a table for comparative purposes; you may want a quick way to drill down further into your list of websites, via tagging, etc.

## The objectives

The objectives are simple:

- Store a list of sites to search.
- Write a routine that will loop through the list and download each html page.
- Search the html page for a keyword. If the keyword is found, find the relating url and store that in a queue.
- Present the user with a list of urls that match the search.
- Use shellexecute to open the url in the browser.

Two tables are required to set up searching. One holds the search category, the other holds all the URLs that will be searched for any given category.

```
Category  -- Parent
        Cat:CatID         Long
        Cat:Description   String(30)

Link  -- Child
        Lin:LinID         Long
        Lin:CatID         Long
        Lin:Description   String(30)
        Lin:Url           String(255)
```

The user will class different searches together. For example, I keep a list of dealerships that sell damaged / repairable vehicles. For a category of Repairables I might have the following Link data:

| **Description** | **Url** |
|---|---|
| Southside Auto | http://www.southside-auto.com/repairables.html |
| Floyd’s Repairables | http://www.floydsrepairables.com/pricelist.htm |
| Interstate Auto | http://www.interstateautocenter.com/listall.cfm |

I also use a global string (`Glo:Search (String 20)`) for the current search keyword. Since I don’t want a case sensitive search, I set the `UPR` attribute.

## How it works

When I select Repairables from my list of categories, the code will cycle through all the child records and fetch the pertinent websites. For simplicity's sake, I've chosen to call a standard Clarion process for reading the child records and calling the appropriate routines. There are plenty of ways to skin this cat, so whether you handcode the loop, use Mike Hansen's SuperStuff to create a custom view, or use a process to handle the child records, is up to you. If you are using Nettalk to download the web page, then you will need to have a window available, though it can be hidden.

I've chosen to use Webfetch.dll to handle the http work of downloading the page. This cool utility is based on the Wininet.dll and is the work of Skip Williams, who described it in Fetching a Web Page With WinInet.

The Webfetch.dll accepts two parameters, the URL that is passed to it and the string that will hold the resultant html data. It traps a number of errors (see Skip's article). Incorporating this into the app is as easy as including the following inside the Global Map:

```
(Module('WebFetch.lib')
   GetaPage(string pURL, *string pTextbuf),long
End
```

Be sure to add WebFetch.lib to your project under Library, Object, and Resource files.

> **Editor's note:** WebFetch is included in the downloadable source zip, but the LIB/DLL is specific to C6.1. You will probably need to rebuild WebFetch for your version of Clarion. Just use Project|Set, load WebFetch.prj, and run Project|Make.

## The ProcessLinks procedure

The `ProcessLinks` procedure will attempt to retrieve and analyze the pages stored in the Links table. It calls Skip's `GetaPage()` function. If `GetaPage` throws an error in trying to retrieve the page, it is probably best not to abort the process, but to simply indicate that there was a problem retrieving the page. Since 0 means a successful retrieve, the following code should do what is needed.

In the Take Record embed of the Process:

```
glo:url = lin:url
glo:page = ''
```

```
If GetaPage( glo:url, glo:page ) > 0
  If glo:url
    GSQ:URL = 'Error fetching ' & clip(glo:url)
    Add(SearchQ)
  End
Else
  Do SearchText
End
```

Once the page is downloaded, three routines are called. The first will search the downloaded html for the keyword. If a match is found, a second will find the URL for the linked page. If necessary, a third routine will assemble the link with the base url. More on that in a minute. Ultimately, the link will be stored in the global queue.

Here's the `SearchText` routine:

```
SearchText  Routine
  DATA
fnd   LONG
  CODE
  StringFoundAt = 1
  x=0
  SL = Len(clip(glo:search))
  FL = glo:search[1]
  LSearch = Upper(clip(glo:search))
  Mx = len(glo:page)
  Fnd = 0
  loop while x < mx
    x+=1
    if upper(glo:page[x]) = FL
      If Upper(glo:page[x : x+(SL - 1)]) = lsearch
        Fnd +=1
        StringFoundAt = x
        loc:tmpstr = glo:page[StringFoundAt : 20]
        glo:url    = lin:url
        Do LoadLink
      End
    End
  End
```

In order to speed things along, the routine searches for a match in the first letter before it looks for the entire string. This probably is immaterial as the bulk of time is taken in downloading the page.

## Grabbing the link

The second routine needs to load the actual URL of the desired page (remember, you’re searching for a link on a page). It would be easy enough to simply indicate that a match was found on the downloaded page and either load the page in a browser or flag the record, indicating the find. This would still mean clicking on the site and manually looking for the link. It’s better, if possible, to locate the link with code.

This is not a foolproof process because not all web pages are created equal. The best thing to do is to load up a sample page, get the source, then search for a keyword. When found, follow the link that leads to the page. Normally this is a matter of finding the closest, prior href tag.

For example, searching for “Prius” might turn up something like the html below.

```
<TR>
<TD><A href="06185.html">06-185</A></TD>
<TD></TD>
<TD>2005 Toyota Prius Hybrid</TD>
<TD>7,397 miles</TD>
<TD>Lf Side Damage</TD>
<TD>$9,500</TD>
</TR>
```

Notice how the link is enclosed inside of the quotation marks, ="06185.html".

Three things need to be done.

1. Find: href=" for the start of the hyperlink
2. Locate the closing: " for the end of the hyperlink
3. Prepend the base url for the site: www.southside-auto.com

The result should be: www.southside-auto.com/06185.html

(Please note that .asp pages generally need to be handled differently than standard html, .cfm, or php pages because of the way they reference the related links. I have been able to work around this in some instances, but there is a lack of consistency among sites. Your best bet is to open the source of the page and examine the link.)

Here’s the code that looks for the link by keyword:

```
LoadLink  Routine
  DATA
i     long
j     long
  CODE
  i=x
    Loop
      i -=1
      loc:href = glo:page[i : 4]
      If LOWER(loc:href) = 'href'
        i +=6  !adding to href = and "
        j = Instring('"', glo:page,1,i)  - 1
        loc:link = glo:page[i : j]
        If ~ Instring('www',loc:link,1,1)
          Do GetBaseURL
          loc:link = BaseURL & '/' & clip(glo:page[i : j])
        End
      GSQ:URL = loc:link
      Add(SearchQ)
      Break
    End
    If i = 1
      Break
    End
    Yield
  End
```

## Getting the base URL

If the link does not contain the entire website address, then it is necessary to append it to the base URL. Usually you have to first find the base URL from the page that was searched.

For example, the page searched is: http://www.southside-auto.com/repairables.html

The link that was found is: 06185.html

To merely append the link to the searched page would produce http://www.southside-auto.com/repairables.html/06185.html which, of course, wouldn’t work.

`GetBaseURL` looks for the double slash and then uses that as a starting point to search for the first single slash to follow. This will result in: www.southside-auto.com/

With Nettalk, there is a command to get the `ClientHost`. This makes assembling the link a snap, because you only need to put the `ClientHost` and your found link together. I was not able to find a similar method in Wininet.dll. It seems that the easiest solution was to search for the double-slash in the url, then find the first single-slash. Backing up one character from the first single-slash should provide the base url.

```
GetBaseURL  Routine
  DATA
DBLSL  LONG
SNGLSL LONG
  CODE
  DBLSL = INSTRING('//',glo:url,1,1)
  If DBLSL
    SNGLSL = Instring('/',glo:url,1,DBLSL+2)
    If SNGLSL
       BaseURL = SUB(glo:url, 1, SNGLSL-1)
    End
  End
```

## Firing Up the Browser

Now that you have the link related to your search term, it’s time to launch the browser via `ShellExecute`.

There are a couple of free templates around for firing up `Shellexecute`, but the example app includes the prototype so you're all set. There are also some paid templates that feature more bells and whistles. A quick search on comp.lang.clarion in Google groups or a visit to Clarion Magazine should get you going on calling `Shellexecute` by yourself.

## Conclusion

On my development machine, with an average DSL connection, I am able to process approximately one link per second. This may not set any speed records, but it is faster than viewing a web page by hand and certainly a lot less cumbersome. With some tweaking this code may do some of the grunt work for you, or at least it will get you thinking of how you could write something similar for your own use.

The downloadable source includes the WebFetch LIB and DLL for C6.1 – for other versions of Clarion you may need to recompile the DLL (the source is included in the zip).

Download the source

---

Jim Albrecht made the mistake of trying Clarion Personal Developer in the late 1980's and has been upgrading ever since. Still primarily a hobbyist, he has written some vertical market apps and does contracting work whenever there's time.

## Reader Comments

Add a comment

Clarion Magazine

# Storing GROUPs in INI Files

**by Jeffrey Slarve**

Published 2006-09-08

I had a need to examine a record buffer and visually compare the contents at the beginning of a procedure with the contents at the end. I decided the easiest way to do this was to write the record buffer contents out to a text file. That way I could use a comparison tool like Beyond Compare to see what data had changed.

I created a function called `GroupToINI`, and called it at the top of `ThisWindow.Init`:

```
GroupToIni(ESTD:Record,'ESTD:Record','c:\temp\estdbefore.ini',1)
```

I then called the function again before the `RETURN` on `TakeCompleted`:

```
GroupToIni(ESTD:Record,'ESTD:Record','c:\temp\estdafter.ini',1)
```

Then I just used Beyond Compare to compare the INI files. Pretty cool.

The code for `GroupToINI` is listed below. The first parameter is the `RECORD` or `GROUP` structure, the second is the name of the INI file section, and the third is the name of the INI file itself. The fourth parameter is a direction byte; you can use this function to read as well as write INI files.

`GroupToINI` aborts if it encounters a null value in the passed structure, so any data beyond the `NULL` field is ignored. When writing data to the INI file, it uses the field name (via `WHO`) as the key.

```
GroupToIni   Procedure(*Group pG,String pSection,|
                       String pINIFile,Byte Direction=1)
```

```
Ndx LONG
A   ANY

  Code

  Ndx = 0
  Loop
    Ndx += 1
    A &= WHAT(pG,Ndx)
    If A &= NULL then break.
    Case Direction
    of 1
      PutINI(pSection,WHO(pG,Ndx),A,pINIFile)
    of 2
      A = GetINI(pSection,WHO(pG,Ndx),,pINIFile)
    end
  end
  A &=NULL
```

You can find a TXA version of this procedure in the downloadable zip. Import it into your application, and make sure the Declare Globally checkbox is checked if you want to use it anywhere.

Download the source

---

Jeff Slarve is an independent software developer and the creator of the critically-acclaimed In Back automated file safeguard utility. Jeff has been a Clarion developer since 1991, and is a member of the group formerly known as Team TopSpeed.

## Reader Comments

Add a comment

Clarion Magazine

# A Template Debugger

## by Russell Eggen

Published 2006-09-06

I've asked for a template debugger for quite some time now and even had some friendly discussions with Bob Zaunere about SoftVelocity producing one. Bob does like the idea as he concedes he would like one as well. He was also quite clear that Clarion has some higher priorities and I can't really argue that point. But dang! I sure wish I had a template debugger! I even had a few ideas about how to make one, but never got past the concept stage.

The subject of template debuggers recently came up on the Internet Relay Chat (IRC) #cw-talk channel.

One of the nice things about IRC rooms is they are open 24/7, and since we have Clarion developers in just about all time zones there are usually a few present. It's great to have some live people to bounce some ideas off. IRC can be a life saver for those who are on site dealing with an emergency, work solo or just plain want some assistance.

Mark Goldberg is a regular on #cw-talk and we often toss stuff back and forth. If his name sounds familiar, it should be. Mark is one of the contributors to the Debuger class (originated by Skip Williams) which is also freely available on my web site.

I mentioned to Mark about my desire to get a template debugger. Mark is a 99.99% hand coder, but he did see the logic of having a template debugger.

Then he dropped a bombshell on me. He said he figured out how to use SysInternals' DebugView, a free system log viewer, with templates, in essence giving template coders a debugger!

It sounded exciting, but I figured it had to be complex. I'm glad I was wrong; after Mark explained his technique, it took me about three minutes to make a template debugger. All you need is Clarion.

### The Ingredients

The magic is a very simple Clarion DLL. Here is the entire source:

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

      CODE

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

That is all there is to it. ODS stands for "Output Debug String". The idea is that you'll call the ODS procedure from the templates (via a neat little `#GROUP`), and the string you pass will appear instantly in the Windows system log, which DebugView displays in a scrolling window.

Your project settings should be like this:

```
    -- Output Debug String for Templates
    #noedit
    #system win32
    #model clarion dll
    #pragma define(maincode=>off)
    #pragma debug(vid=>full)
    #pragma optimize(cpu=>386)
    #compile "ods.clw"
    #link "C:\Clarion6\BIN\ods.dll"
```

Save this as a .PR or .PRJ file (a text project file). Please note the last line. If that is not your Clarion bin location you will need to change it. And you need an export map in the same folder specified in the #link line above:

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

That is the entire DLL source you need. Load up the .PR project file and compile the DLL; verify that the DLL is in your Clarion bin folder.

## Adding ODS Messages to Templates

Next, you need to add a small #GROUP to your templates. I called mine ODS as well. Here's the source:

```
#GROUP(%ODS,%Debug)
#RUNDLL('ODS.DLL','ODS','DebugTrue symbol is now: ' & %Debug),WIN32
```

#GROUP statements are like template procedures, and the first line of the #GROUP is like a procedure prototype. The first parameter is the name of the #GROUP. The second parameter is a string value, passed by value. This lets me pass expressions as well as variables.

#RUNDLL calls my DLL, which is listed in the first parameter. The second is the public procedure in my DLL and the third is the parameter passed to my procedure.

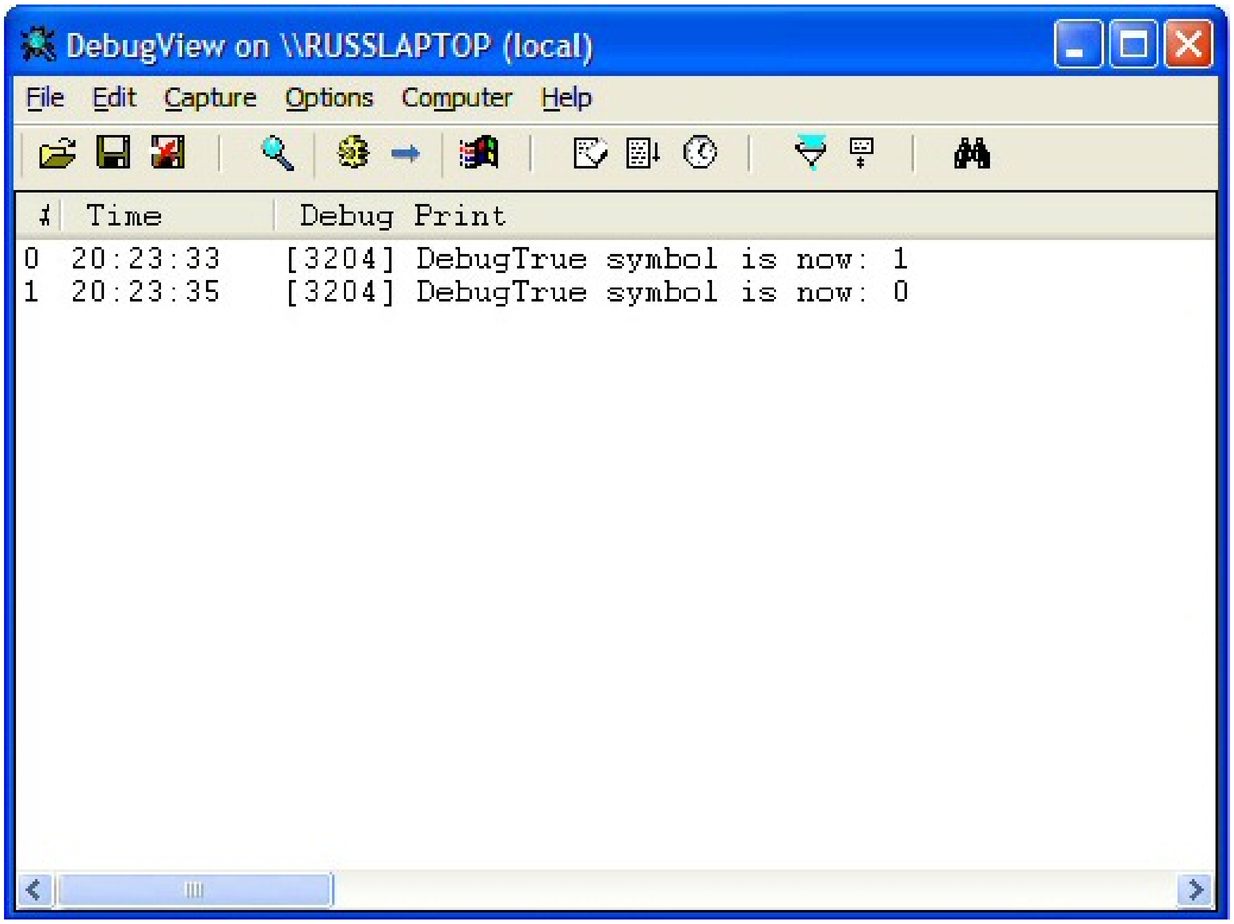
I needed to give this a test, but where to start? I decided on a #PROMPT set up as a check box. That should be easy. But why #PROMPT? Because it has this really neat attribute, WHENACCEPTED(), which triggers whenever the developer changes a value in the templates.

I decided to use my Debuger template as a test subject. Below is the code for my checkbox that turns the Debuger on or off for a given application (line break added):

```
#PROMPT('Activate Debuger class ?',CHECK),%DebugTrue,DEFAULT(%True),
  WHENACCEPTED(%ODS('DebugTrue set to ' & %DebugTrue)
```

This single line of code (wrapped for readability) displays a check box with text. The value is stored in the symbol %DebugTrue. The WHENACCEPTED line is, in this case, like the Accepted event "embed" for this "control". When I check or uncheck the box, the AppGen will call my %ODS #GROUP.

So when I fire up DebugView, I see this after ticking the checkbox twice:


DebugView on \\RUSSLAPTOP (local)
File Edit Capture Options Computer Help
# Time Debug Print

```
0   20:23:33   [3204] DebugTrue symbol is now: 1
1   20:23:35   [3204] DebugTrue symbol is now: 0
```

**Figure 1. Template debugging in action**

## Conclusion

This is just a very simple demonstration of how you can use this DLL and DebugView to debug your templates. For example, you could examine the data inside `#FOR` loops using `#FIX(%MySymbol,%OriginalSymbol)` and other such constructs. Just add a line of code like this:

```
    #INSERT(%ODS,'Symbol value is ' & MySymbol)
```

Or you can just output some text like this:

```
    #INSERT(%ODS,'My debug message here')
```

If you modify the shipping templates keep in mind that upgrades may cause you to lose your changes.

---

Russ Eggen has been using Clarion since 1986. Until about 1996, he was using it for business applications, mostly accounting programs. Afterwards he joined Topspeed as a consultant, and later as an instructor. He was a founding member of SoftVelocity when that company formed from Topspeed in

May 2000. He left SoftVelocity in January 2001 and now works for his own company, RadFusion Inc. He still teaches and lectures, and is currently working on a new book and setting up a local Clarion classroom. Russ enjoys flying, scuba, and applied philosophy, and with great effort you might coax him into political discussions.

## Reader Comments

Add a comment

- » Russ, Thank you for your kind words, and prominent...
- » Mark, Quite correct. I leave such uses to the template...

Clarion Magazine

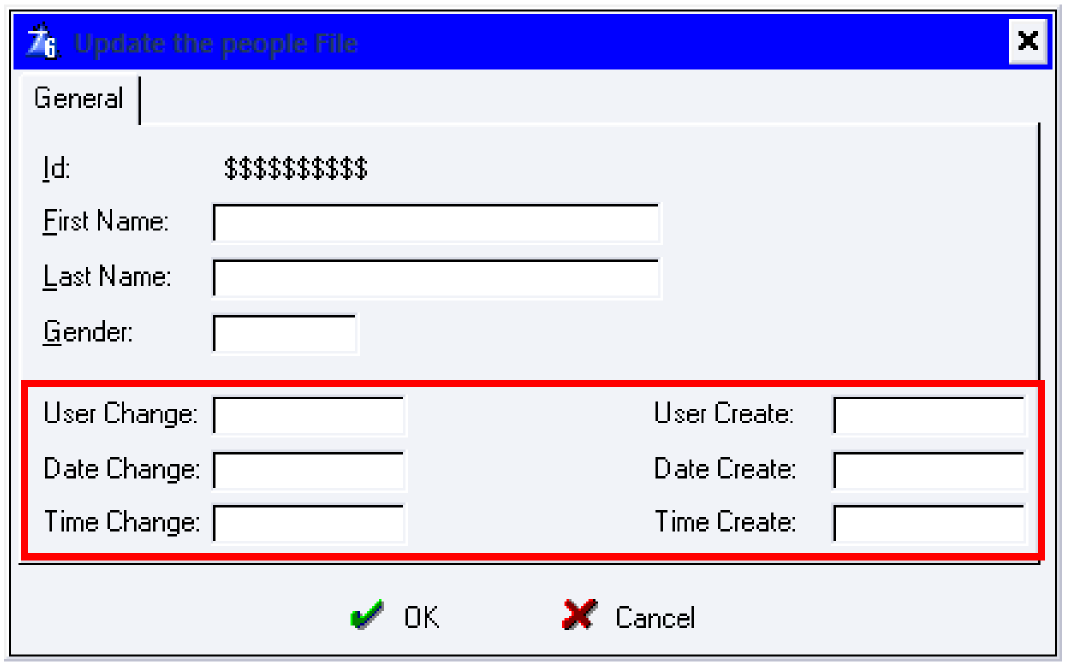
# An Economical Record Status Control

**by Nardus Swanevelder**

Published 2006-08-28

On a couple of my latest projects I kept on running out of screen "space". What I mean by this is that I just have too many prompts and fields to populate and you can only add so many tabs. I started to look at my screen design to see where I could save space.

I saw that I used the following structure, made up of six read-only fields, on all of my update screens:


Update the people File

General

Id: $$$$$$$$$$

First Name:

Last Name:

Gender:

User Change: User Create:

Date Change: Date Create:

Time Change: Time Create:

OK Cancel

**Figure 1. Normal method for displaying the User Create and User Change Status**

You can clearly see that the area in the red rectangle takes up a lot of space. I decided to see if I could create one entry field to display all of this information on one line.

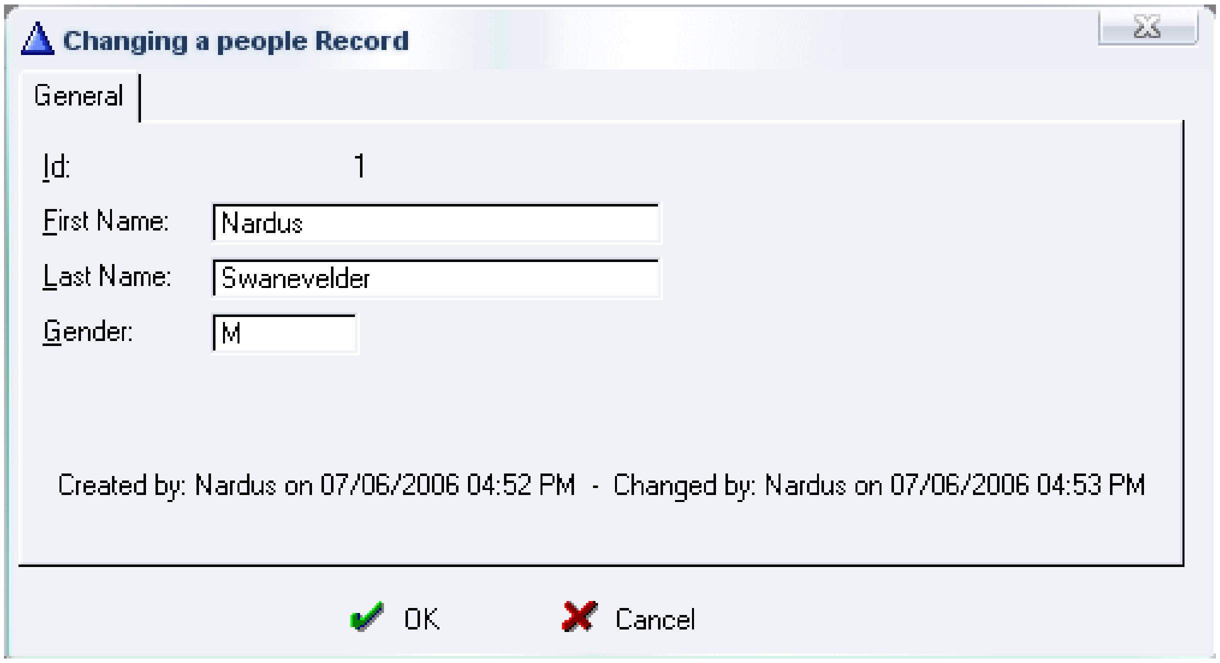
I wanted the entry field to be populated based on certain rules. In pseudo code the implementation of these rules looks like this:

```
If UserCreate <> '' and UserCreate <> ''
   Control = 'Created by John on 16/01/2005 ' |
     & '- Changed by Sean on 23/05/2006'
ElsIf UserCreate <> '' and UserCreate = ''
   Control = 'Created by John on 16/01/2005'
ElsIf UserCreate = '' and UserCreate = <>
   Control = 'Changed by Sean on 23/05/2006'
Else
   Control = ''
End
Display(Control)
```

The Clarion Code looks like this:

```
If PEO:UserCreate <> '' and PEO:UserChange <> ''
  LOC:RecordCreatedChanged = 'Created by: ' |
    & Clip(PEO:UserCreate) & ' on ' |
    & format(PEO:DateCreate,@d17) & ' ' |
    & format(PEO:TimeCreate,@t7) & '  -  ' |
    & 'Changed by: ' & Clip(PEO:UserChange) |
    & ' on ' & format(PEO:DateChange,@d17) |
    & ' ' & format(PEO:TimeChange,@t7)
Elsif PEO:UserCreate <> '' and PEO:UserChange = ''
  LOC:RecordCreatedChanged = 'Created by: ' |
    & Clip(PEO:UserCreate) & ' on ' |
    & format(PEO:DateCreate,@d17) |
    & ' ' & format(PEO:TimeCreate,@t7)
Elsif PEO:UserCreate = '' and PEO:UserChange <> ''
  LOC:RecordCreatedChanged = 'Created on ' |
    & format(PEO:DateCreate,@d17) & ' ' |
    & format(PEO:TimeCreate,@t7) & '  -  ' |
    & 'Changed by: ' & Clip(PEO:UserChange) |
    & ' on ' & format(PEO:DateChange,@d17) |
    & ' ' & format(PEO:TimeChange,@t7)
Else
  LOC:RecordCreatedChanged = ''
End
```

This is a lot of code to type on every update form. Time for a template! But first, see Figure 2 for an example of what the screen will look like after the new method is implemented.



Id: 1
First Name: Nardus
Last Name: Swanevelder
Gender: M

Created by: Nardus on 07/06/2006 04:52 PM - Changed by: Nardus on 07/06/2006 04:53 PM

**Figure 2. New Method for displaying the User Create and User Change Status**

## The template

The first question that needs to be answered is what kind of template to create? You could use a code template but then you would have to create a local variable on each update form, you would have to populate that control on the screen, and then you would have to remember to add the code template. Those are too many things to remember; a much better choice is a control template.

The Clarion help defines a control template as follows:

> "Control templates (#CONTROL) place a related set (one or more) of controls on a procedure's window and generate the executable source code into the procedure's embed points to provide the controls' standard functionality."

The control part of the template is easy as I only want an entry field that will display the information:

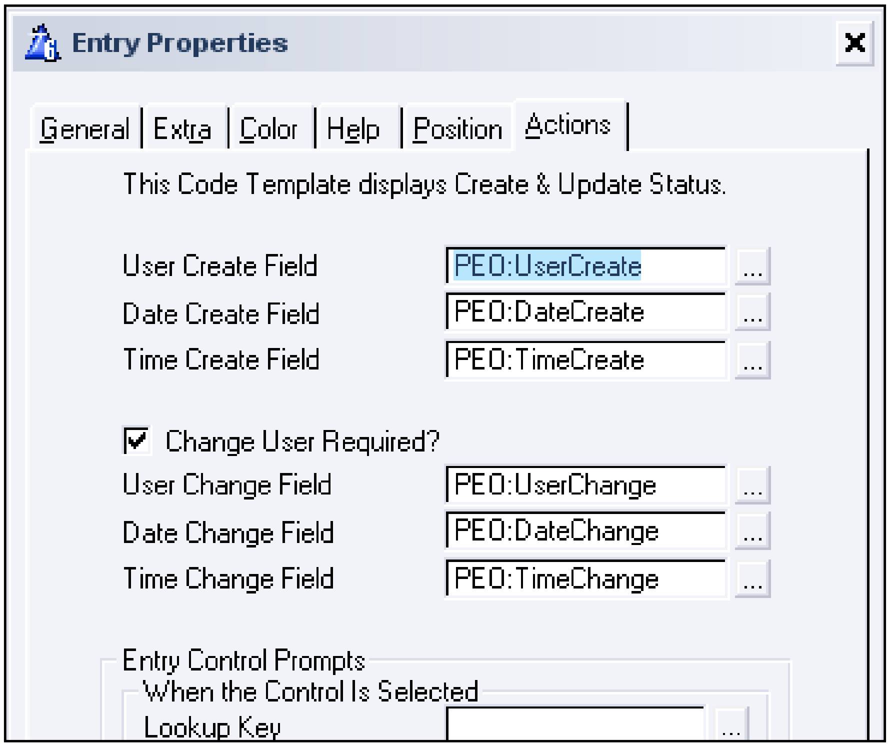
```
CONTROLS      ENTRY(@s200),AT(,,290,10),USE(LOC:RecordCreatedChanged)
                , SKIP, TRN, CENTER, READONLY
END
```

I want the entry to be read-only, transparent, skipped and it must center the text. The control should also be at least 290 DLUs wide.

I need to know which table fields are the Create and the Change fields, and therefore I populated my control with the following prompts:

```
#PROMPT  ('User Create Field',FIELD),%UserCreateField,REQ
#PROMPT  ('Date Create Field',FIELD),%DateCreateField,REQ
#PROMPT  ('Time Create Field',FIELD),%TimeCreateField,REQ
#DISPLAY ('')
#PROMPT  ('User Change Field',FIELD),%UserChangeField,REQ
#PROMPT  ('Date Change Field',FIELD),%DateChangeField,REQ
#PROMPT  ('Time Change Field',FIELD),%TimeChangeField,REQ
```

Figure 3 shows the template properties as displayed in the Window Formatter.



This Code Template displays Create & Update Status.

| | |
|---|---|
| User Create Field | PEO:UserCreate |
| Date Create Field | PEO:DateCreate |
| Time Create Field | PEO:TimeCreate |

☑ Change User Required?

| | |
|---|---|
| User Change Field | PEO:UserChange |
| Date Change Field | PEO:DateChange |
| Time Change Field | PEO:TimeChange |

Entry Control Prompts
When the Control Is Selected
Lookup Key

**Figure 3. Example of populated Control Template properties**

Figure 4 shows the appearance of the control in the Window Formatter.

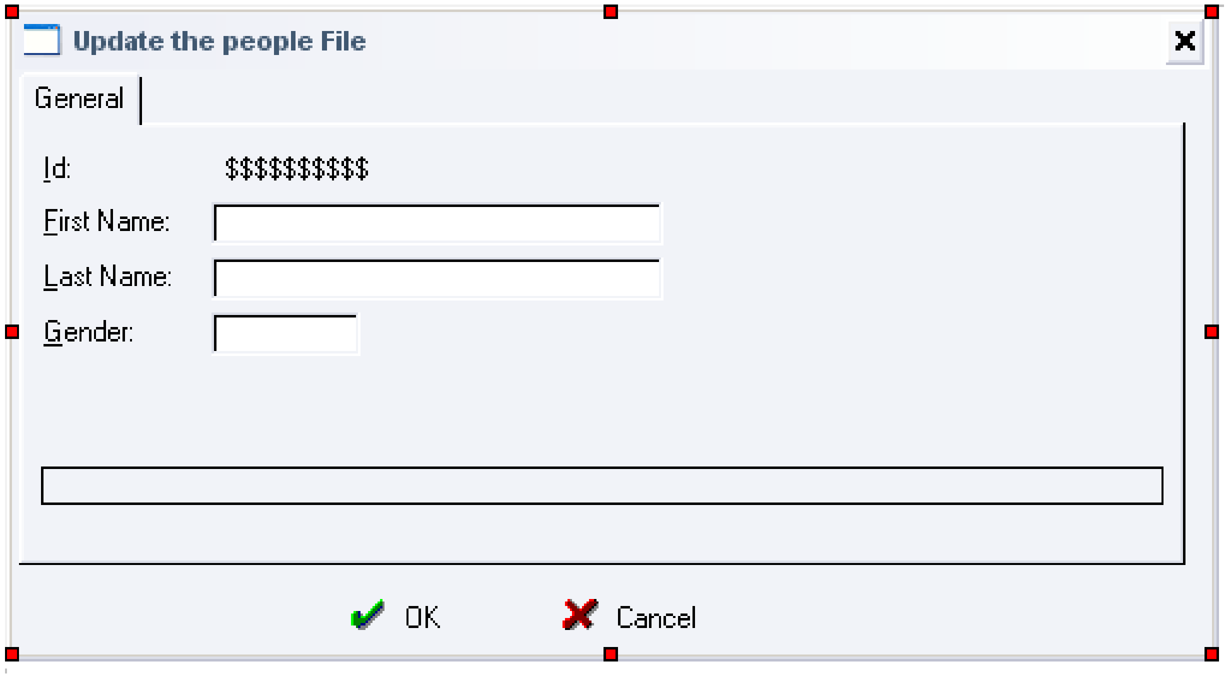


**Figure 4. Screen after the control template has been populated**

Next I have to define a local variable that will hold the display string:

```
#AT(%DataSection),PRIORITY(4000)
LOC:RecordCreatedChanged    String(200)
#ENDAT
```

I generate the display code into the `WindowManager`'s `Init` method, as follows:

```
#AT(%WindowManagerMethodCodeSection,'INIT'),PRIORITY(8001)
If Self.Request <> InsertRecord
  #If(%ChangeUserREQ)
  If %UserCreateField <> '' and %UserChangeField <> ''
    LOC:RecordCreatedChanged = 'Created by: ' |
      & Clip(%UserCreateField) & ' on ' |
      & format(%DateCreateField,@d17) & ' ' |
      & format(%TimeCreateField,@t7) & '  -  ' |
      & 'Changed by: ' & Clip(%UserChangeField) |
      & ' on ' & format(%DateChangeField,@d17) |
      & ' ' & format(%TimeChangeField,@t7)
  Elsif %UserCreateField <> '' and %UserChangeField = ''
    LOC:RecordCreatedChanged = 'Created by: ' |
      & Clip(%UserCreateField) & ' on ' |
      & format(%DateCreateField,@d17) & ' ' |
      & format(%TimeCreateField,@t7)
  Elsif %UserCreateField = '' and %UserChangeField <> ''
    LOC:RecordCreatedChanged = 'Created on ' |
      & format(%DateCreateField,@d17) & ' ' |
      & format(%TimeCreateField,@t7) & '  -  ' |
      & 'Changed by: ' & Clip(%UserChangeField) |
      & ' on ' & format(%DateChangeField,@d17) |
      & ' ' & format(%TimeChangeField,@t7)
  Else
    LOC:RecordCreatedChanged = ''
  End
  #ELSE
  If %UserCreateField <> ''
    LOC:RecordCreatedChanged = 'Created by: ' |
      & Clip(%UserCreateField) & ' on ' |
      & format(%DateCreateField,@d17) & ' ' |
      & format(%TimeCreateField,@t7)
   Else
      LOC:RecordCreatedChanged = ''
   End
  #ENDIF
End
#ENDAT
```

Note that I used the standard MS Windows short date and time formats, `@d17` and `@t7`. If you are not happy with this you will have to change the template to use the format that you want or change the template to ask what format it should use for the date and time.

As you can see in the above code, I only build the display string if `Self.Request` is not equal to `InsertRecord`. There is no point in building this string on an Insert as both the Create and Change information should be empty. I have also added an option to the code template where you can specify that a specific control does not have Change information. I found that in certain instances I allow a user to create a record but not to change it and therefore I had no Change fields to populate in my template.

## Features to be added

There are several ways this template can be enhanced. You can change the entry control to a `PROMPT` or a `TEXT` field so that it will be possible for the text to wrap. You can also modify the template to allow for multiple instances on a single form.

## Using the template

Adding the template to an update form is easy:

- Register the template, which you can find in the attached source code
- Open your application and open the procedure where you want to add the new code template

- Click on Populate, Control Template and choose the DC Create/Change Status-Control template
- Position the cursor where you want the control and click with the mouse
- Complete the control's properties, as shown in Figure 3.

## Summary

By combining six read-only fields into a single string, I saved a lot of space on my update forms. And by creating a control template, I made it easy to populate this string with the right data. Control templates are relatively easy to write, and best of all, if you need to make changes to the code, you only need to change the template to have your changes appear throughout your application.

Download the source

---

Nardus Swanevelder was born and raised in South Africa. He was a networking engineer for seven years before he moved over to the commercial side of the business. Nardus has developed a Sale Cycle Management system for the Information and Communication Technology industry. He has been programming in Clarion since 1989, and holds B.Com and MBA degrees. In his spare time Nardus lectures Financial Management to B. Com Hons students at North-West University.

## Reader Comments

Add a comment

- » Nice idea. Perhaps you could save even more screen real...

Clarion Magazine

# Commentary on UKCUG, Clarion Roadmap

**by Dave Harms**

Published 2006-08-21

In Richard Rose's report on the UK CUG meeting with Bob Zaunere and others from SoftVelocity I mentioned that I'd be posting a few more of Richard's screen shots. Since I wasn't at the meeting, the associated commentary below is largely my own semi-informed conjecture. For the most part I've cropped the pics down to some bits I found particularly interesting.

Figure 1 shows a portion of the new IDE including the Tools palette. The area on the left is a sheet - each major item, such as ASCII Table, is a tab. The Windows Forms tab is selected, and underneath the tab header you can see a list of standard WinForms controls. Below that is a tab for Data (sources, I presume), then Components (that is, components without a visual aspect), and Custom (visual) Components. No big surprised there. But don't overlook the upper tabs.

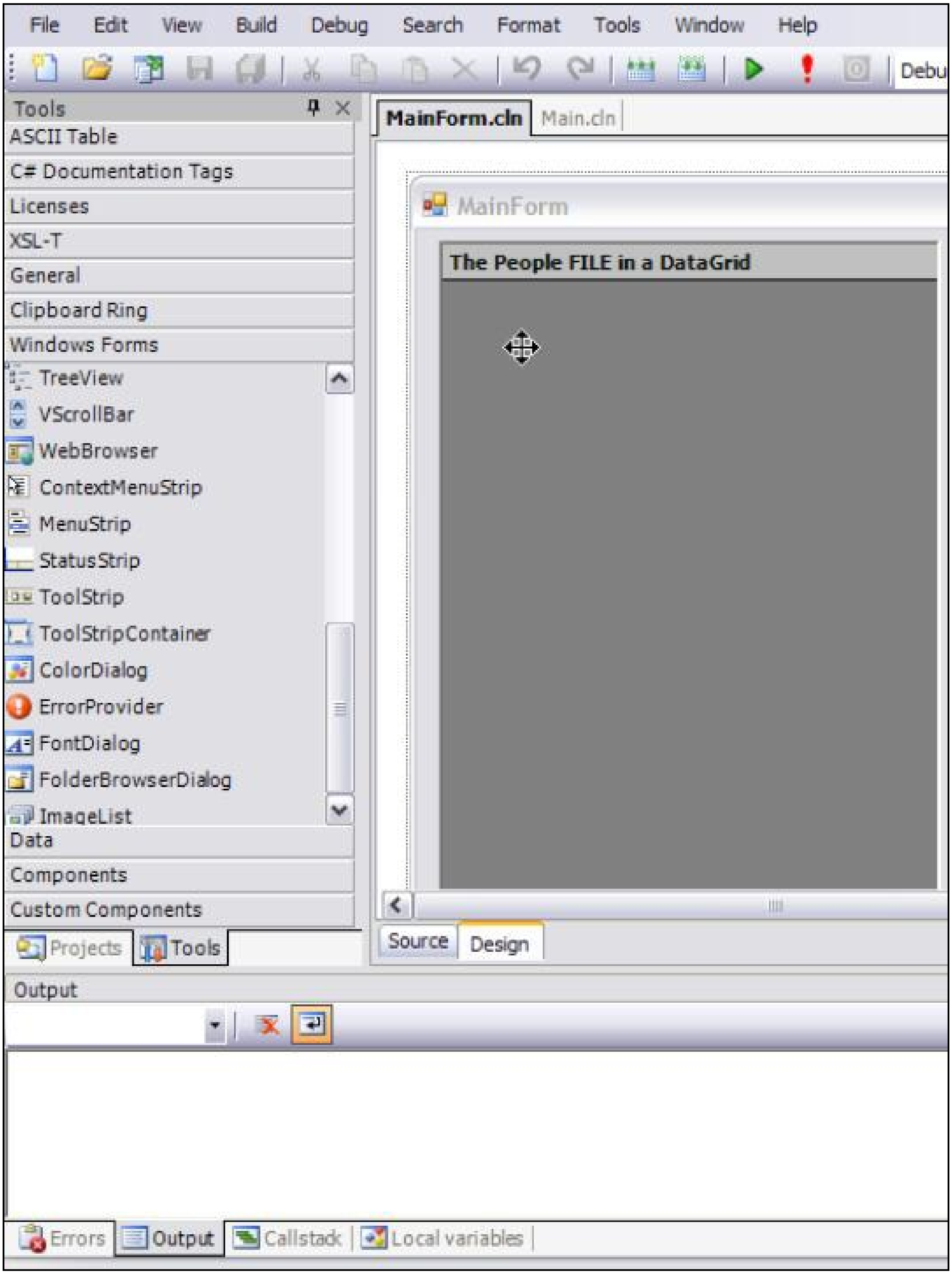

File
Edit
View
Build
Debug
Search
Format
Tools
Window
Help
Debu
Tools
ASCII Table
C# Documentation Tags
Licenses
XSL-T
General
Clipboard Ring
Windows Forms
TreeView
VScrollBar
WebBrowser
ContextMenuStrip
MenuStrip
StatusStrip
ToolStrip
ToolStripContainer
ColorDialog
ErrorProvider
FontDialog
FolderBrowserDialog
ImageList
Data
Components
Custom Components
Projects
Tools
MainForm.cln
Main.cln
MainForm
The People FILE in a DataGrid
Source
Design
Output
Errors
Output
Callstack
Local variables

**Figure 1. The Tools palette**

Click on ASCII table and you'll see a list of ASCII characters and their numeric codes. C# Documentation Tags is a list of XML tags used for documenting C# code (it would be nice if Clarion.NET includes support for these tags, which make it easier to generate source code docs). To use a tag, just drag it into your code. Most of these are single line tags, except for list items. Dragging list-number into your source code causes the following text to appear:

```
/// <list type="number">
/// <listheader>
///     <term></term>
///     <description></description>
/// </listheader>
/// <item>
///     <term></term>
///     <description></description>
/// </item>
/// </list>
```

The Licenses header lets you drag license text into your source; in the regular version of #develop, these include the BSD, GPL, and LGPL licenses. The Clipboard Ring keeps the last 20 or so items copied to the clipboard, and if you want to paste any one you just drag it back into your code.

I'm not sure what the General tab does in the Clarion IDE, but as it's easy to create your own tabs it could be just about anything. You can add custom controls of various kinds to your own tabs (or some of the other tabs), and you can also drag blocks of text to your own tabs. This is going to be a very handy feature for anyone who does any embed or hand coding: think of all those snippets of code you're forever repeating because you never get around to putting them into a template....

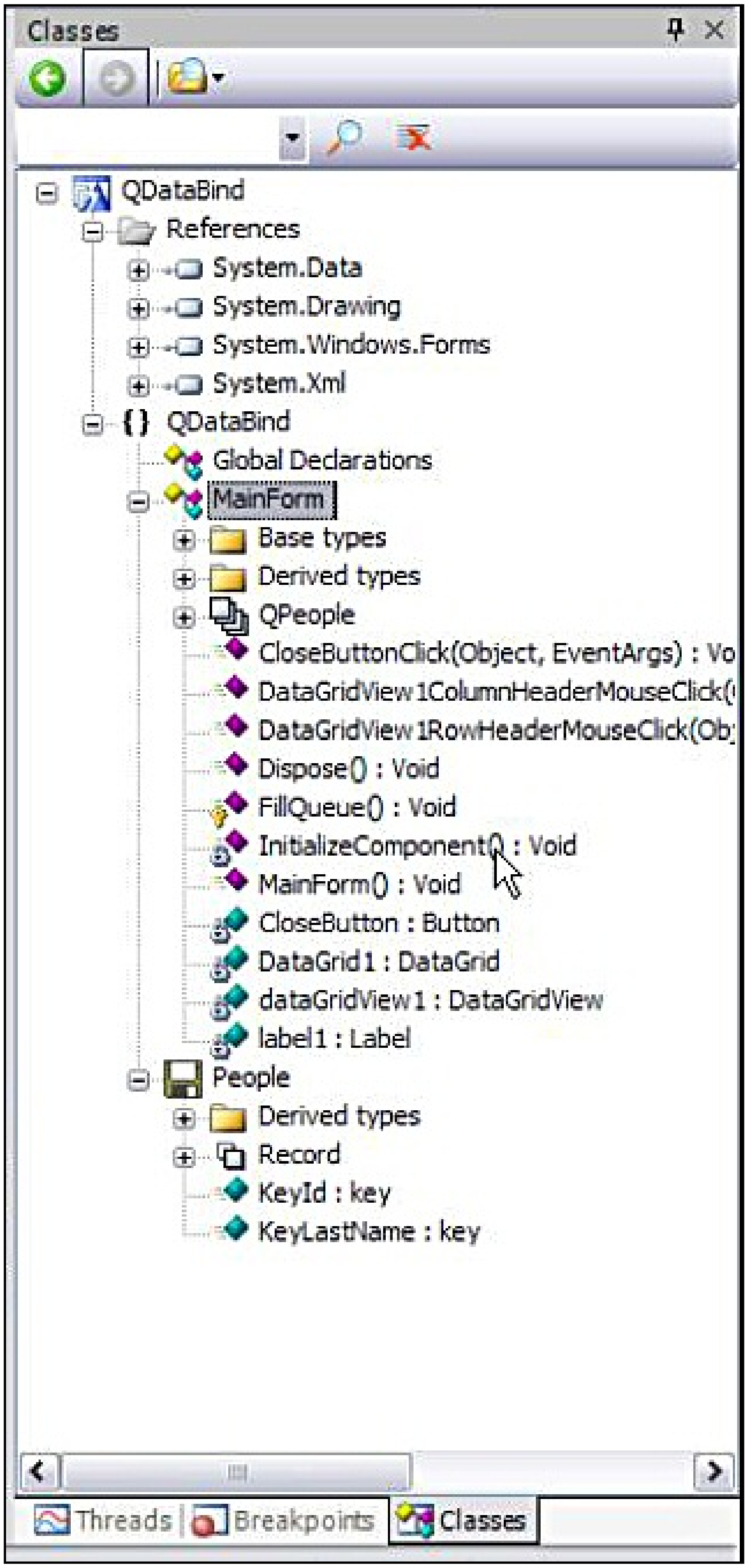
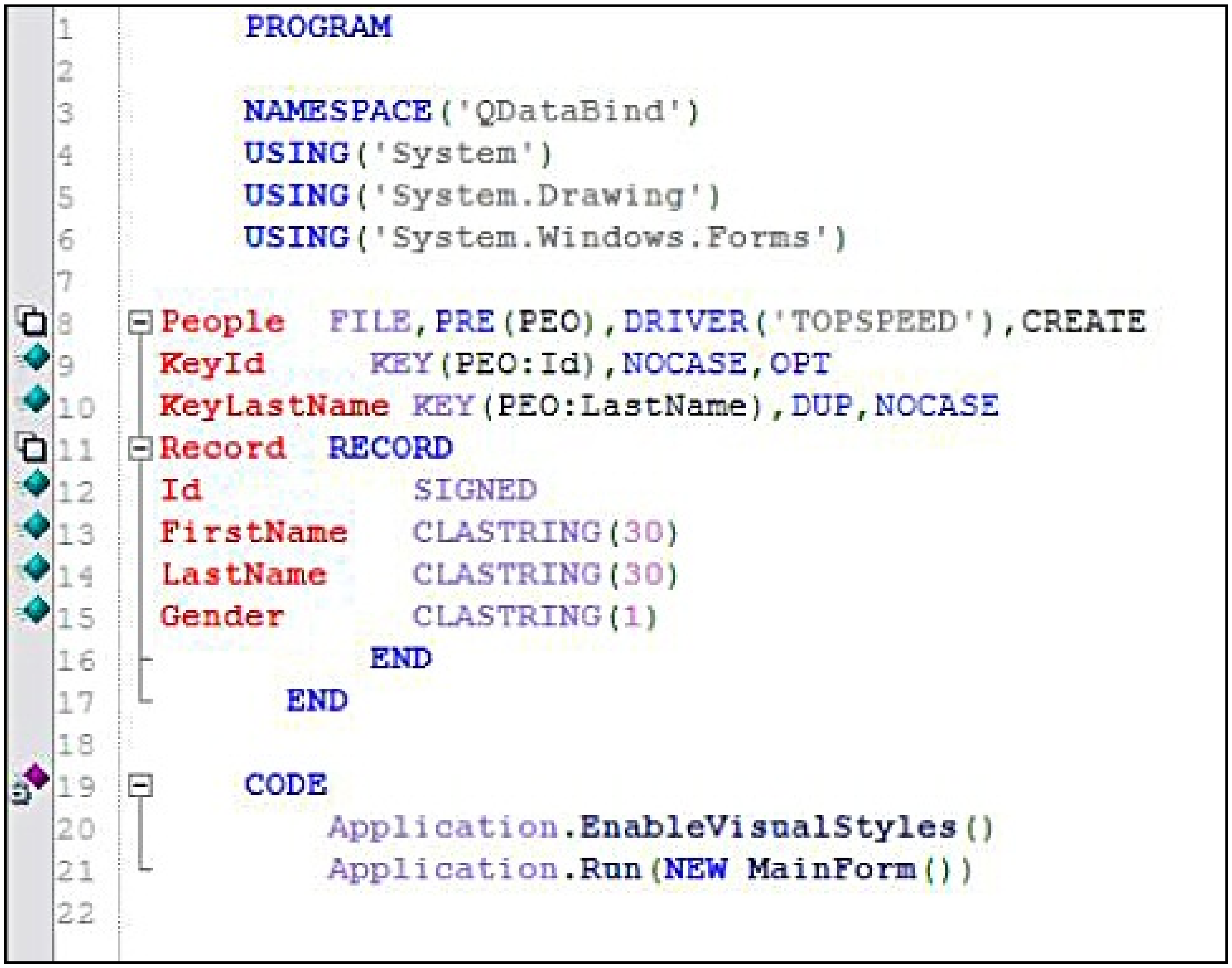


**Figure 2. The class browser**

Figure 2 shows the IDE's class browser. As with most newer IDEs, the browser is in a toolbox that is (or can be) always open. And unlike Clarion's ABC class browser, which has to be popped up in its own application-modal window, this class browser is also a navigation tool. Note that Figure 2 shows a data source, the People TPS file, in the tree as well.



```
         PROGRAM

         NAMESPACE('QDataBind')
         USING('System')
         USING('System.Drawing')
         USING('System.Windows.Forms')

People   FILE,PRE(PEO),DRIVER('TOPSPEED'),CREATE
KeyId      KEY(PEO:Id),NOCASE,OPT
KeyLastName KEY(PEO:LastName),DUP,NOCASE
Record   RECORD
Id           SIGNED
FirstName    CLASTRING(30)
LastName     CLASTRING(30)
Gender       CLASTRING(1)
           END
         END

         CODE
           Application.EnableVisualStyles()
           Application.Run(NEW MainForm())
```

**Figure 3. Starting a .NET application**

Figure 3 shows the first few lines of a small Clarion.NET application. Note the NAMESPACE and USING directives. Namespaces are (very) roughly analogous to prefixes in Clarion. Let's say you have two variables called `count`, one local, the other global. The compiler will flag the conflict and tell you it's only going to use one of them. But if you call them GLO:count and LOC:count you won't have a problem. Similarly, namespaces let you assign what are essentially prefixes to classes. If you don't have any naming conflicts, you can use the class names without prefixes, and tell the compiler via the USING directive which namespaces it should search.

For more on .NET namespaces see my article .NET Basics: What Is .NET, And Why

Should I Care? Part 2.

Note also that the PEOPLE record declaration uses CLASTRING instead of STRING. In .NET, even though there seem to be simple data types like int, under the hood, everthing's a class, even `int` (which is just an alias for the System.Int32 class).

Strings in .NET are quite different animals from Clarion strings. The System.String class has numerous methods that let you handle tasks like changing case, trimming spaces, concatenating, splitting, searching, string slicing, etc. String is a powerful class, and offers many advantages over the Clarion STRING data type. But converting all your Clarion string handling to .NET string handling is potentially a lot of work, so Clarion.NET has a CLASTRING type, which lets you continue to use your old Clarion string handling code.

So why not leave STRING as is, and call the .NET string class something like NetString? Changing the meaning of STRING is going to break a lot of code. Bruce Johnson has pointed out that change will be painful, but

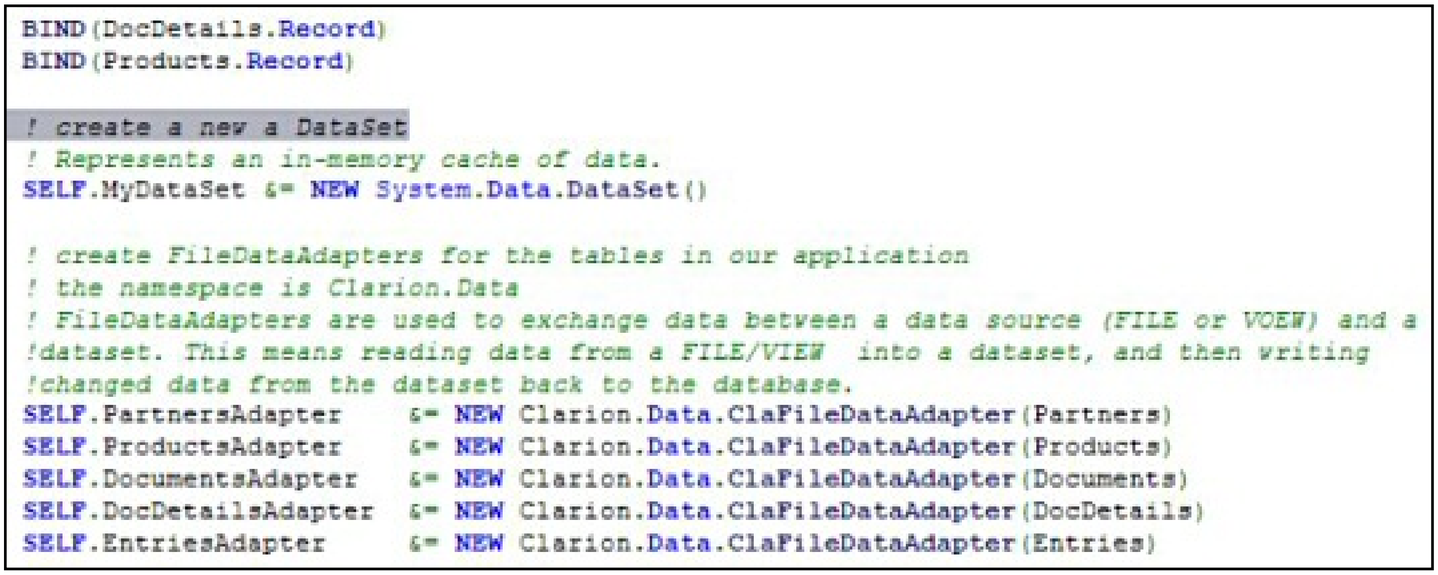
> it will make it easier in the long run for Clarion.Net programmers to converse with other .Net programmers, and will make documenting Clarion code (to be used by other .Net languages) easier. In other words when Clarion programmers use a STRING, and document any variable as a STRING, then all .Net programmers (regardless of language) will understand exactly what we mean. We'll be talking a "common language".

Finally, in Figure 3, note the two lines after the CODE statement. Application is really System.Windows.Forms.Application, which provides code to start, stop, and otherwise manage WinForms applications. There's no instance of Application, and there doesn't need to be, which is something that sounds confusing at first.

In the Clarion (4,5,6,7) world, there are two ways to declare a class: with the TYPE attribute, and without the TYPE attribute. If the class declaration uses TYPE, then you have to create an instance (or multiple instances) of the class before you can use it. If you don't use the TYPE attribute, you use the class as declared (in essence, the class and the object are one and the same). This is the default.

In many languages, including C#, the default declaration is the equivalent of a Clarion TYPEd class. If you want to use the class without having to declare at least one instance, you have two choices. You can give the class the static modifier, or you can give individual methods the static modifier. When you're used to a class in one or the other state (TYPEd/not static or unTYPEd/static) it can be confusing to see a class that is only partly unTYPEd/static. But it happens all the time in C# and other programming languages. An application may have a static main method (the automatically-called entry point for console applications) and one of the first things that main method often does is create an instance of its own class. What better way to get up and running without the burden of allocating a bunch of extra memory at compile time?

The code in Figure 3 doesn't do anything that fancy, but it does pass a new instance of MainForm to the Application's static Run method. All of Application's exposed methods are static, so in essence Application is a static class.



```
BIND(DocDetails.Record)
BIND(Products.Record)

! create a new a DataSet
! Represents an in-memory cache of data.
SELF.MyDataSet &= NEW System.Data.DataSet()

! create FileDataAdapters for the tables in our application
! the namespace is Clarion.Data
! FileDataAdapters are used to exchange data between a data source (FILE or VOEW) and a
!dataset. This means reading data from a FILE/VIEW  into a dataset, and then writing
!changed data from the dataset back to the database.
SELF.PartnersAdapter    &= NEW Clarion.Data.ClaFileDataAdapter(Partners)
SELF.ProductsAdapter    &= NEW Clarion.Data.ClaFileDataAdapter(Products)
SELF.DocumentsAdapter   &= NEW Clarion.Data.ClaFileDataAdapter(Documents)
SELF.DocDetailsAdapter  &= NEW Clarion.Data.ClaFileDataAdapter(DocDetails)
SELF.EntriesAdapter     &= NEW Clarion.Data.ClaFileDataAdapter(Entries)
```

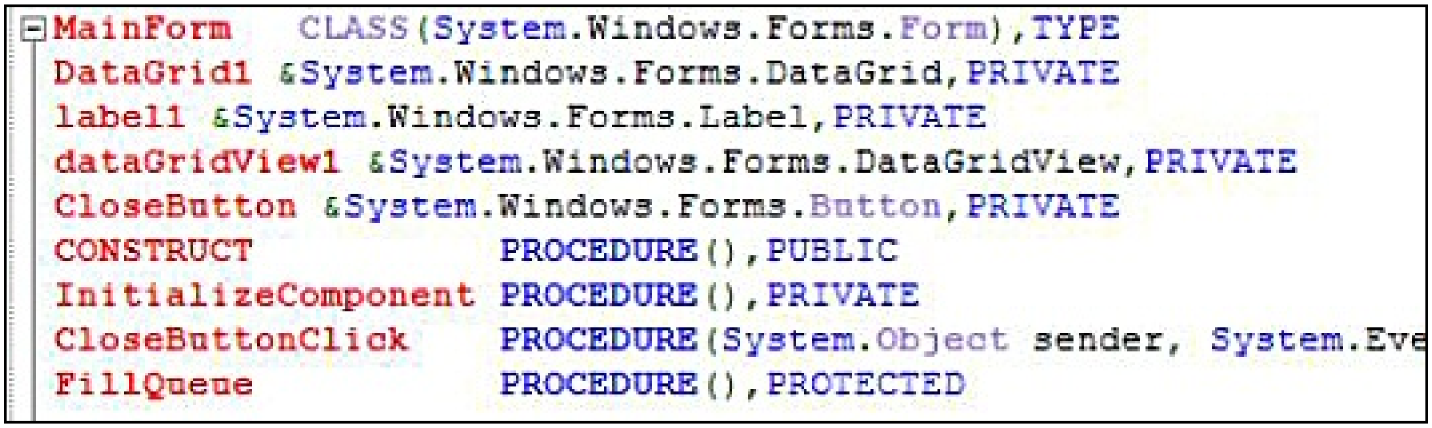
**Figure 4. FileDataAdapters**

Figure 4 shows the use of the FileDataAdapter class. By way of clarification, here's an excerpt from a blog reply by Bob Z:

> The application architecture generated by the templates divides your application into layers; Presentation layer, Business logic layer (rules), and Data access layer. The templates allow you to make the decision on what type of data access layer your want to use. You could choose to have your data access layer generated using the familiar Clarion data access model. All of the Clarion data entities (FILE,RECORD,VIEW,QUEUE and GROUP) support data binding to any .Net control that provides for data binding. There are some obvious advantages to this model (and some not so obvious advantages). Or you can choose to have your data access layer generated to utilize the ADO.Net object model. We have DataAdapters for both FILE and VIEW structures that plug right into the ADO.Net object model, so you

can happily work with what you know or the new ADO.Net objects. In the future you'll have the option to generate the data access layer utilizing an ORM model like nHibernate or Gentle.Net, or when its released using LINQ.

What Figure 4 shows, then, is how you would plug a TPS file into ADO.NET. And ADO.NET is just one of the ways you can display data in a WinForms control - you can also directly bind files, views, queues, and groups to controls as you do now in Clarion for Win32 (though the mechanisms will probably be a bit different). .



```
MainForm    CLASS(System.Windows.Forms.Form),TYPE
DataGrid1  &System.Windows.Forms.DataGrid,PRIVATE
label1  &System.Windows.Forms.Label,PRIVATE
dataGridView1  &System.Windows.Forms.DataGridView,PRIVATE
CloseButton  &System.Windows.Forms.Button,PRIVATE
CONSTRUCT                  PROCEDURE(),PUBLIC
InitializeComponent        PROCEDURE(),PRIVATE
CloseButtonClick           PROCEDURE(System.Object sender, System.Eve
FillQueue                  PROCEDURE(),PROTECTED
```

**Figure 5. The new window structure.**

The Clarion WINDOW structure is one of the few major changes in Clarion.NET - it's completely gone (along with the ACCEPT loop), replaced by a class declaration based on System.Windows.Forms.Form. The templates will, of course, take care of generating the class, and Bob has blogged that there will be a migration tool to assist with the conversion of hand coded WINDOW structures.

## Road map comments

SoftVelocity recently released its road map for C7 and Clarion.NET. Note the proviso that "As always, these plans are subject to change." But it's still very nice to have something in print.

Funky code names aside, it's worth noting that SoftVelocity expects to release both C7 and Clarion.NET, both including dictionary and appgen, this year. Given that neither product is in beta yet, and there are only a little more than four months left, I don't think a gold release is in the cards for this year. Still, any release at all would be most welcome. I'm actually a little surprised to see the suggestion that Clarion.NET, with templates, could be out at approximately the same time as C7, since Clarion.NET represents a complete platform shift. Time will tell.

And I'm still hoping there's at least a limited release of the Clarion.NET hand coder's edition, which some observers at the UK CUG meeting seemed to think was usable now.

Notable plans for next year (and beyond) include 64 bit support for Clarion Windows (non-.NET) applications. The 64 bit barrier would not appear to be much of a concern for Clarion.NET - floating point calculations seem to be the only real migration issue.

The first release of Clarion.NET will be labelled version 2.0, to keep the numbers in sync with the .NET platform release. The next major release, for .NET 3.0, will presumably be called Clarion.NET 3.0. That release should have support for Microsoft's next big user interface step, Windows Presentation Foundation (WPF - formerly Avalon). You can see some examples of what WPF can do at seewindowsvista.com.

SoftVelocity plans to offer a high level API to Indigo, Microsoft's new web services platform.

And for those Clarion developers who want to develop for mobile platforms, the second release of Clarion.NET, Bob says "We intend to have complete IDE design surfaces for .NET Compact Frameworks up and running".

Also planned is support for LINQ, a very cool data query technology from Delphi/C# architect Anders Hejlsberg that lets you use a single syntax to extract and work with data from a variety of different sources, including SQL and XML. I've played with the LINQ preview in C# - it's pretty slick, and will be a real boon to anyone who's had to work extensively with XML in Clarion.

## TPS backward compatibility

One other issue that's come up recently is a misunderstanding about TPS backward compatibility. In both C7 and Clarion.NET you will have the *option* to use Windows encryption providers to encrypt TPS files. If you take that approach, your TPS files will not be readable by earlier versions of Clarion. If you do not take that approach, your TPS files will be backward compatible.

---

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 ClarionMag Blog

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

XML All blog entries
XML All new items, including blogs

## Blog Categories

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

## July/Aug PDF

Direct link

Posted Tuesday, August 22, 2006 by Dave Harms

If you're wondering why the July PDF isn't up yet, it's because July and August are a combined issue. Once per year I do this to free up some time for our family vacation. It isn't always July/August - last year it was June/July and the year before August/September. This year the ClarionMag summer break also gave me some time to attend OSCON, the O'Reilly Open Source Convention, where I followed the database track. Clarion Magazine's coverage of that conference will be included in the July/August PDF.

I have a number of articles in hand for September, and I'm looking forward to increased coverage of Clarion 7 and Clarion.NET as we approach first release.

## SpaceMonger 2.1!

Direct link

Posted Wednesday, August 16, 2006 by Dave Harms

SpaceMonger 2.1, the long-awaited successor to version 1.4, has been released! I'm a huge fan of SpaceMonger for disk cleanup; SM's visual map makes it easy to locate those bloated files/directories you no longer need.

Some new features of note:

- pie and bar charts of your data, in addition to the traditional treemap
- progressive display - SM builds the data view as it scans
- option to hide directories and files - very handy if you have some big items you want removed from the view so the smaller items are more visible

As of this release SpaceMonger is a free trial product - after 30 days, you'll need a $19.95 license, which is still one of the best deals around. And I'm not aware of any product that does SpaceMonger's job as well.

[HT: Leroy Schulz]

Clarion Magazine

# The ClarionMag Blog

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

XML All blog entries
XML All new items, including blogs

## Blog Categories

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

## Summer break, PDFs, new articles, and the big sale!

Direct link

Posted Tuesday, September 05, 2006 by Dave Harms

You may have noticed that I've posted the July PDF, which was supposed to be the July/August PDF. I normally schedule a break in the publishing schedule sometime in the summer and combine two months into one. This allows some time for family vacation (and we had another great road trip!).

When I went to do the July/August PDF I discovered I had well over a month's worth of material. So I've revised the schedule a little. The July PDF contains the July articles plus the OSCON articles which were published in August; as a result the July PDF is of typical size.

August and September will now be the combined issue instead of July/August, and this is actually a big help as we have a family emergency to take care of. My sister-in-law was seriously injured in a freak boating accident a few days ago, and we'll be heading west shortly to help out.

I do have a number of interesting articles in hand, however. The first two will most likely be Russ Eggen's template debugger code, which uses Mark Goldberg's technique, and a nifty web search app by Jim Albrecht. Also look for a cool coding tip from Jeff Slarve. I'll try to have at least two of these up this week, but given our travel schedule it may take until early next week.

And don't forget our terrific DevCon or No DevCon Sale! (Check out the alphabetical article list.) This is fantastic pricing, and it's probably the last time I'll be able to offer such low prices, given current exchange rates (ClarionMag is in Canada, but we charge in US dollars). Sign up, re-up, do what it takes! You won't regret it.