Clarion Magazine

Clarion News

- » NetTalk 4.17 Released
- » BackFlash Demo
- » NetWeb demo on Oak Park Solutions Server
- » Whitemarsh Newsletter for October, 2006
- » SealSoft Releases xReportPreview 3.3
- » NetTalk Web Demo Updated
- » Clarion Scripting Language
- » Free BoTpl Templates
- » ClarionTools NEW! Query Wizard v6.05 Released
- » J-Fax 1.30 Gold
- » SetupBuilder 5.6 Build 1659
- » UltraTree Sale Extended
- » UKCUG Clarion Presentations Online
- » Clarion Developers .COM Domain Special Extended and Expanded
- » FullRecord 1.70
- » Clarion Desktop Updated
- » GenWise Documentation Released
- » EasyCOMCreator 1.05
- » Mallorca Icon Special
- » Scandinavian Clarion Newsgroup
- » Emailing SealSoft Company
- » Smooth Scrolling Control 1.0
- » UltraTree 10th Anniversary Sale
- » ImageEx 3.6
- » SetupBuilder 5.6 Build 1648

- » TList7 ActiveX Clarion Wrapper
- » FreeImage Project 3.9.1.1
- » Firebird 2.0 RC5
- » Gitano Distributors Wanted
- » Ingasoftplus Forum Returns
- » J-Skype News
- » Webmaster Icon Collection
- » Virtual and Dedicated Server Sale for Clarion Developers
- » Whitemarsh Newsletter for September 2006

[More news]

Latest Free Content

- » Welcome To The New Server!

[More free articles]

Clarion Sites

Podcast

[Track lists, more podcasts]

Clarion Blogs

Latest Subscriber Content

Signing Your Applications, Part 1

Setupbuilder 5 gives you the ability to sign your program installers. But increasingly Windows security makes it advisable to be able to sign your actual software products as well. Jane Fleming covers the basics of digital certificates in preparation for signing applications. Part 1 of 2.

Posted Tuesday, October 31, 2006

Clarion 101: Describing your Data

One of the reasons for Clarion's popularity is the way it handles data. Bruce Johnson begins a series for Clarion 101 on designing databases and creating data dictionaries.

Posted Friday, October 27, 2006

Recursive Adds

"Heads down data entry" makes the ability to add multiple records without having to return to a browse desirable. Even customers in lighter duty situations want it. Dr. Parker shows how to deliver the goods.

Posted Thursday, October 26, 2006

Welcome To The New Server!

Welcome to the new ClarionMag web server! We're still testing the new install and the old server will remain up for a while yet. If you encounter any problems on this server let me know. But please note that during this testing phase the new server may occasionally be unavailable for brief periods of time.

Dave Harms, Publisher

Posted Thursday, October 26, 2006

Classes For Background Processes

C6 introduced true threading, and threads are a great way to handle background tasks. But threads seem very procedural - you start some code running, and it runs until it ends. What if you want to use threads in an object-oriented environment? Is it even possible to have a class that runs on its own thread? How do you control threads in an object-oriented environment? David Harms demonstrates one approach to threading classes.

Posted Friday, October 20, 2006

Throwing Users Out: Methods of Computation

Having determined, in the previous episode, how to throw a user out of a window, Steve Parker returns to the problem of knowing when to throw the user out.

Posted Tuesday, October 17, 2006

Clarion 101: Choosing a Database

Before you can build an application, you need a database. Clarion is amazingly flexible when it comes to database support; flat file, relational, client side code, server side code, SQL, non SQL, it's all there. So how do you decide on a database?

Posted Thursday, October 12, 2006

Replicating IDLE: Throwing Users Out

"When the user hasn't done anything for x minutes (or seconds), I want to close the window. How can I do this?" This question comes up fairly frequently. As is so often the case using Clarion, there are a number of ways to accomplish the goal of throwing the user out.

Posted Thursday, October 12, 2006

Clarion 101: Understanding the IDE, Part 2

Dave Harms concludes his overview of the IDE with a look at the Clarion language, the ABC class library, the database grammar and file drivers, and the browse/form concept.

Posted Thursday, October 05, 2006

PDF for August/September 2006

All articles for August/September 2006 in PDF format

Posted Tuesday, October 03, 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:

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

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

From The Publisher

About Clarion Magazine

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

Subscriptions

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

Satisfaction Guaranteed

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

Dave Harms

ISSN

Clarion Magazine's ISSN

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

About ISSN

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

Clarion Magazine

Clarion News

Search the news archive

AppScanner Lite Debugger

AppScanner is a 'lite' debugger. It can hook into a running process and show information about: Loading/unloading of DLL/OCX; Starting/exiting of processes and threads; Debug information send from process; Process memory usage; Exceptions encountered in the process (first/second chance). All information can be saved to disk.
Posted Friday, November 10, 2006

Clarion Desktop 1.35 Adds Downloads, Video

Clarion Desktop 1.35 adds two new sections on the home page, "Latest Downloads from Par2.com", and "Clarion Audio / Video Library". The Par2 section shows links to the latest downloads from Steve's Par2.com site, and the Audio / Video section lists links to any audio / video material on Clarion (including things like podcasts, video presentations, etc). You can also now view "All Products"! "My Products" still shows what you have installed and what new updates are available for download etc, but "All Products" displays all the Clarion Desktop Compatible 3rdParty products (89 as of today). There are now "buy now" links next to products that you don't already own. As of version 1.30 Clarion Desktop also stores your installation passwords.
Posted Friday, November 10, 2006

IAlchemy Office Schedule

Pratik Patel will be undergoing a corneal transplant on Nov.28. He doesn't anticipate being out of commision for more than two days, but advises customers to plan their bugs accordingly.

Posted Friday, November 10, 2006

ProImage Demo and Sale Price Extension

The ProImage demo is available for immediate download and ProImage sale prices are. Until Monday, November 13, 2006 you can still purchase ProImage at the special price of $199.95 (you save $50.00) or get the ProImage/ProScan bundle for only $374.95 (you save $74.95). Both the "Classic" ProImage Editor interface and the new "PowerToolbar" version of the ProImage editor are shown and you can easily switch between the two. The demo is an interactive program with multiple buttons that call the ProImage editor in different modes of operation. It illustrates some common applications of ProImage as well as the reusability of the editor within the same application without recoding. Also included is a wizard that uses the ProImage Image Assembler to build a composite photo ID with data, a photo and a scanned signature. Note: The PowerToolbar edition of the editor is code complete and will ship in the next update to ProImage. Also included in the release will be all the imaging icons needed for the ProImage procedures in 10 different color/styles. These icons are provided free courtesy of www.icons-icons.
Posted Friday, November 10, 2006

J-Fax 1.32

J-Fax 1.32 is now available. This build includes a fix for legacy apps using the Icetips Previewer, as well as a new feature whereby portrait reports are now automatically faxed in portrait-mode.
Posted Friday, November 10, 2006

J-Skype 1.10

J-Skype 1.10 includes an important fix to the GetContacts functionality. With J-Skype you can now: Chat with other Skype users from your Clarion APPs; Call other users over Skype from Clarion; Use Skype as a network layer between two Clarion APPs; Retrieve contact lists from Skype (All users, online users, etc); Set Skype properties (Mood text, Skype name, etc). $49 during the beta, the price goes up on gold release.
Posted Friday, November 10, 2006

SysInternals Now At Microsoft

SysInternals was recently purchased by Microsoft; you can now find the free utilities on Microsoft's web site.

Posted Friday, November 10, 2006

SQLShell Released

Comsoft7 & KwikSYSTEMShave released an SQL version of the Duke shell products. This release works with MS-SQL 7 or newer, and SQL scripts are provided. Firebird and MySQL versions are in the works.
Posted Friday, November 10, 2006

Oak Park Wins Hosting Award

Oak Park Solutions has been selected by the HostReview's editorial staff to be one of the "Top10" Hosts in the new Fastest Growing Company category for November 2006. The Top 10 Best Web Hosting Awards are based on the overall product offering, value, customer service and users' reviews of the selected companies. To celebrate, Oak Park Solutions is running specials on many web hosting products for Clarion developers and their clients, starting with hosting accounts as low as $3.99 per month.
Posted Friday, November 10, 2006

Icetips Previewer 2.4

Icetips Previewer 2.4 has been released and is available for download. This new release contains a new wizard to create and maintain Icetips Previewer procedures and a new help file for all templates - directly accessible from the Clarion IDE - as well as updated documentation. This release also adds compatibility with Clarion Desktop, and contains fixes to various small bugs here and there, including maintaining report settings when printing to a selected printer, such as orientation, paper size etc
Posted Friday, November 10, 2006

Data Interoperability Community of Interest Handbook Released

Whitemarsh announces the availability of the Data Interoperability Community of Interest Handbook. This book is available at a special prepublication price of $49.95, $30 off the retail price. Book Abstract: Data-centric organizations engineer "data" to support databases and information systems. Regrettably, many of the databases and information systems that have been built over the past 50 years have been "Interoperability Challenged." Some are even level "minus 3" on a 1-5 point Software Engineering Institute scale. In response, organizations demand amended ways: Produce interoperable databases and information systems by yesterday if not sooner. Great pronouncements are made;

strategies are set into place; Power Point presentations are produced at prodigious rates; and if lucky, funding follows. Surface validity of these pronouncements and approaches abound everywhere. Over the years, however, instead of doing the hard work of building interoperable databases and information systems, efforts are redirected onto the tracks of the newest, fastest traveling silver bullet. All too often, however, objectives are not achieved, and sometimes, project failure or death is the result. This book is not about another silver bullet. Rather, this book is how to achieve data interoperability the old-fashioned way: Earned. It's accomplished by engineering metadata repository environments, having clear missions, functions, organizations, and such, and then building, through consensus, interoperable data and process specifications, one at a time, within communities of interest. Once implemented, more expansive specifications can be created by intersecting shared data across communities of interest. Ultimately, the objective, shared data is achieved. The key components of this approach have been tried before, and succeeded. This book is modeled on these successes. The book's approach also works top-down or bottom-up. That is, through Forward and/or Reverse engineering. This book sets all this out in a step by step manner. It builds on already known and proven concepts. It provides the blueprint of the metadata infrastructure, the interoperable product specifications, and the work breakdown structures. This is the handbook for the data interoperability community of interest team members.
Posted Friday, November 10, 2006

ProDomus Adds Clarion Desktop Support

ProDomus has added Clarion Desktop support and is providing a file you can download that will list all PD products. All current ProDomus installs have been changed to be Clarion 3rd Party Association compliant. ProDomus using a modified version of the accini install to allow changes to allow finding App References in the Clarion Accessories menu of items that may have been translated - a suggested change that has not yet been officially incorporated.
Posted Friday, November 10, 2006

ProDomus PD Version Notes and CLE

Working Smarter - PD Version Notes and Command Line Editor Version 6.4 0001 is now available. The PD Command Line and Version Notes Utility (PDCL) is many programs in one, all designed to help you work smarter and faster. Version and Note Log. You record what you've done, when you did it, and in what version. If support or maintenance questions arise, you can quickly research exactly when something was done using the simple text file created. You include the file created in your application and automatically update the version and build number. This is run from the Clarion Accessories menu or

from Programs to Execute in the project editor. Environment Menu Editor. You can modify your Clarion working environment to work smarter by using PDCL's line item entry editor. Clarion Menu. If you want to add a favorite tool to the Clarion menu, it can be done quickly from the Windows Run dialog using the EDIT parameter. Menu items you add can run built-in Clarion Applets (selectable from a drop list), a program you identify, or programs associated with specific file extensions. File Dialogs. You can easily add and move file masks in the Clarion open file dialog and specify associated programs. If you have a project that works with templates or a specific set of source files, you can set up and position file type mask for these. You can also run programs like MS Word or your favorite help file editor. Dialog Order. You can easily change the order of items on the accessories menu so that those that you use more frequently are at the top. Fix Inoperable Items. If important accessory items have become inoperable because menu capacity has been exceeded, they can easily be moved up and others removed. Translate the Environment. If you want to translate items in the Clarion environment or in the PDCL utility itself, you can do this in the PDCL Edit window (Translator Plus in built-in). Clarion Shortcut. Start Clarion with different class library Sets using PDCL and the path the class library as a parameter. In Multi DLL project where you want to use a limited set of class library files, you can copy the files to a separate directory and start Clarion with a pointer to these as the location of the ABC libraries. This eliminates unused class files from your application. It also allows you to modify files without changing the originals or having them replaced by a new install. PDCL was first released in 1999. This version adds the editor interface for editing the Clarion environment. It is available on ClarionShop or by direct order from ProDomus.
Posted Friday, November 10, 2006

SV Driver Sale Reminder

IP Driver, In-Memory and Dynamic File drivers are on sale until November 22. Save $125 off any driver bundle or save $75 off any single driver. Purchase the drivers now and you'll receive the Clarion 7 upgrade versions of the drivers for free.
Posted Wednesday, November 08, 2006

Huenuleufu RSS Feed

The Huenuleufu site now has an RSS feed. Press the XML graphic on the "news" section, or manually set the address to rss.xml on your RSS client. Automatic detection is enabled in the main page for programs like Google Desktop.
Posted Thursday, November 02, 2006

PostgreSQL Support in FM3

File Manager 3 (aka FM3) builds basic automatic file management into your program. This means you can make changes to your dictionary file structure, and FM3 will take care of the file structure changes automatically. It forms part of your application. FM3 supports data conversion between file drivers as well. FM3 supports a number of databases, the most recent addition being PostgreSQL. You can now implement the full functionality of FM3 into your PostgreSQL applications. Another new addition is the ability to connect to Oracle Express (the freeware version of the Oracle's Flagship product).
Posted Thursday, November 02, 2006

CapeSoft License Amnesty in November

CapeSoft's license policy requires one license per developer. To help you come right and return to that clean slate that you started out on, CapeSoft is making the month of November 2006 amnesty month. Anyone who needs to purchase additional licenses, can purchase these at 50% of the normal purchase price. Note: This is only for purchasing additional licenses - you must have at least one license for those products that you are purchasing additional licenses for. You're welcome to make the most of the special to purchase new accessories at the same time - but the first license will cost the normal price and from there on the discount will apply. Amnesty month ends on November 30, 2006. If you are purchasing additional licenses, then please indicate this in the comments field of the online order form as follows: 50% discount for additional licenses during Amnesty month.
Posted Thursday, November 02, 2006

Clarion ProImage Released

ProImage is a programmers image editor that lets you add advanced image processing to your applications in minutes. ProImage's user interface will adapt itself depending on how you call it from your app. You can call it to simply allow the user to obtain and process an image, passing it back to you when finished, or as a stand alone image editor, or as a page editor for other controls (such as ProScan). ProImage can process images from files (with a picture dialog), drag and drop from Windows Explorer, paste from the Windows clipboard (either a photo or even an Internet URL), from a scanner, a direct load from Internet URL and from a WebCam. ProImage has a unique Visual Targeting System that provides your users with a complete realtime visual representation of what they are getting as they process the image. It maintains a high quality output because it uses a virtual coordinate system to create output images from the original - not from a reduced screen image. ProImage produces high quality print images that print the correct size and with the

DPI you request. It also produces highly optimized PDF output of images that print at the correct size and with the quality level (DPI) that you set. These are not the typical standard report bloated PDF files, but have very small file sizes. ProImage can also produce entire photo sets at the same time (for example a thumbnail, screen image and a high quality print image). ProImage normally sells for $249.95, but until Nov 8 2006 you can buy ProImage for $199.
Posted Thursday, November 02, 2006

NetTalk 4.17 Released

NetTalk 4.17 (beta) is now available. NetTalk 4 allows you to build stand-alone web servers using Clarion. The program is written much like any Clarion program, in the Clarion language, using templates. It is deployed just like any Windows program, as an Exe with DLLs etc. There are no dependencies on IIS or Apache (or anything else). Because NetTalk's web server is a Clarion program it can use any data source that Clarion can use, including flat-file systems like TPS. NetTalk also includes classes for doing many other Network functions, including email send & receive (SMTP/POP3) , ftp clients, web clients, peer-to-peer communications, remote program shut-downs and much more. This version incorporates a number of new features including: XP-Taskpanels; XP Style Tabs; Child Browses; Bug fixes. As of November 1 the price for the NetTalk upgrade from version 3 to version 4 will increase from $149 to $199. The current NetTalk price for new owners is $399.
Posted Thursday, October 26, 2006

BackFlash Demo

A Turbodemo of BackFlash is now available. BackFlash customers, please feel free to distribute this demo to your customers as part of their product training.
Posted Thursday, October 26, 2006

NetWeb demo on Oak Park Solutions Server

Dean Burgess over at Oak Park Solutions, is offering Virtual Machine Hosting from $34.95 per month, 50GB disk space, 2000 GB bandwidth, 3 fixed IP addresses and full Admin Access. In order to test the suitability of this for NetTalk Web Server customers, Dean has given CapeSoft a free server for testing. You can find the NetWeb demo there.
Posted Thursday, October 26, 2006

Whitemarsh Newsletter for October, 2006

The Data Interoperability Community of Interest Handbook was finished. Then several late reviews arrived with some excellent suggestions. Rather than wait for a 2nd Edition of the book, these additions were incorporated. The table of contents and the first chapter have both been regenerated and are again posted for review. These materials are located at: www.Wiscorp.com/dicoihb.pdf . The book has now been sent to the printers. A pre-publication sale announcement will occur in two weeks. There will then be an official Announcement Sale for the book, and this notice will be sent out in the early part of November. The second item is version 6.09 of the Metabase CASE/Repository system. This version is almost complete and expected to be released in early November. This version has many small enhancements. Most importantly, this version now enables the printing of hierarchies and networks of data through SQL/ODBC-based report writers such as Crystal Reports. Also released with this version will be a set of data conversion instructions and a program for those Metabase users who wish to upgrade. Email assistance is also available regarding the database conversion for all current Metabase licensed users. The third item is the latest edition to the Whitemarsh Short Paper Series: "Modeling Data & Designing Databases". This paper takes the position that these topics are two very different processes. The paper includes a case study in which an organization first attempted to design a database without previously modeling the data. The end result was vastly inadequate. The paper then sets out a high level set of process steps that are thoroughly documented in Whitemarsh materials. The total net time of including these steps, by first modeling the data, is ultimately negative. By first thoroughly modeling the data, a completely sound database design is achieved much faster. Together, the combined processes of data modeling and database design is far faster and cheaper than iterating through cycles of database design, application implementation, re-design, data conversion, reprogramming, and redeployment. Additionally, all the modeled data's metadata is then available for future modeling and design with subsequent cycles becoming even shorter, faster, and cheaper. This short paper is located at: www.wiscorp.com/sp/sp06.pdf
Posted Thursday, October 26, 2006

SealSoft Releases xReportPreview 3.3

SealSoft's xReportPreview 3.3 includes a configurable preview window in a separate thread, new printer dialog, design- and run-time page footers and headers, saving of reports in WMF files, and more. xReportPreview is $59.
Posted Thursday, October 26, 2006

NetTalk Web Demo Updated

In anticipation of the 4.17 NetTalk release, Bruce Johnson is looking for some traffic on

the NetTalk web server demo
Posted Thursday, October 26, 2006

Clarion Scripting Language

ClaScript includes a script language and a window formatter. Features include file access, use of BINDed functions, control structures, and more. Clarion-like syntax.
Posted Thursday, October 26, 2006

Free BoTpl Templates

The free BoTpl template suite includes the BoAppInfo, DicPrint, and EmbsAllOut Utility templates .
Posted Thursday, October 26, 2006

ClarionTools NEW! Query Wizard v6.05 Released

ClarionTools is happy to announce the general availability of the new and improved Query Wizard Version 6.05. This release is available immediately and is free to all registered Query Wizard 6 developers. If you are an existing customer your are encouraged to update your products to this latest version, if you haven't seen Query Wizard in a while, please download the latest demo from the website at www.clariontools.com to see why it is a favorite of Clarion developers worldwide. Changes in this release include: Language translations and template support for Dutch and Spanish; Template and help updates to clarify multiple first generation child queries; Fixed IP driver problem.
Posted Thursday, October 26, 2006

J-Fax 1.30 Gold

J-Fax 1.30 Gold has been released. J-Fax can now add "Fax" to the report previewer's "File" menu (Clarion 5.5 and 6). You can either pass it a fax number (from say a Customers Browse), or it will prompt you for a fax number at runtime. Template options, examples, and documentation have all been improved. The fax server can now be configured via either variables, template settings, or an INI file. You can use J-Fax with: Standard Clarion 5.5 and 6 reports; C6 Advanced Report Generators (ReportToPDF, ReportToXML, ReportToFax); Icetips Previewer; Fomin Report Builder; CPCS. Fax Server source code is also available. This lets you route all faxes on a network to a single computer, which will then send all faxes out via its own fax modem. Client computers don't even need to have MS Fax installed.
Posted Friday, October 20, 2006

SetupBuilder 5.6 Build 1659

SetupBuilder 5.6 build 1659 has been released. As always you may use the "Check for Updates" option within SetupBuilder to auto-update your current SetupBuilder 5.x version. You can also download the full SetupBuilder 5.6 install image from the web site.
Posted Friday, October 20, 2006

UltraTree Sale Extended

The UltraTree 10th anniversary sale has been extended three days, ending with the close of business Monday, October 23, 2006.
Posted Friday, October 20, 2006

Clarion Magazine

Signing Your Applications, Part 1

by Jane Fleming

Published 2006-10-31

Who do you think you are? Why should I trust you?

Of course, we're all friends here. (Right?) But the Internet – and, increasingly, the whole world of computing – is potentially dangerous. As Microsoft and others scramble to protect us, it's only a matter of time until your users start seeing ominous "unverified publisher" warnings such as that shown in Figure 1.

Figure 1. Unknown Publisher security warning

Setupbuilder 5 gives you the ability to sign your program installers. But increasing Windows security makes it advisable to be able to sign your actual software products as

well.

In this article, I'll go over the basics of digital certificates, then walk you through signing an individual piece of software. Since signing is something you do to a compiled program, using Microsoft's tools, you'll need to go through the process each time you ship a new version. (You can also sign DLLs, CAB files, and MSI files with the tools I'll be describing.) I'll also cover a way of automating the signing process.

Caveat: as with many other aspects of my life, this is an area where I know enough to be dangerous but don't claim great expertise.

What is a digital certificate?

Basically, a digital certificate is a group of bytes that serves to verify one's identity. But in all common computer uses, that's only a part of what this data does.

Many people are first exposed to digital certificates (whether they know it or not) when they start purchasing items online. When you go to the check out with your purchases, usually the website address changes. Instead of beginning with the familiar **http://**, it now begins with **https://**. The **S** stands for *secure*.

Beyond that change in the address bar, you may have noticed a browser icon that appears when you have a secure connection. With Internet Explorer, it looks like a padlock.

Figure 2. Browser certificate icon indicating a secure connection

If you double-click the padlock, you'll see the site's certificate information

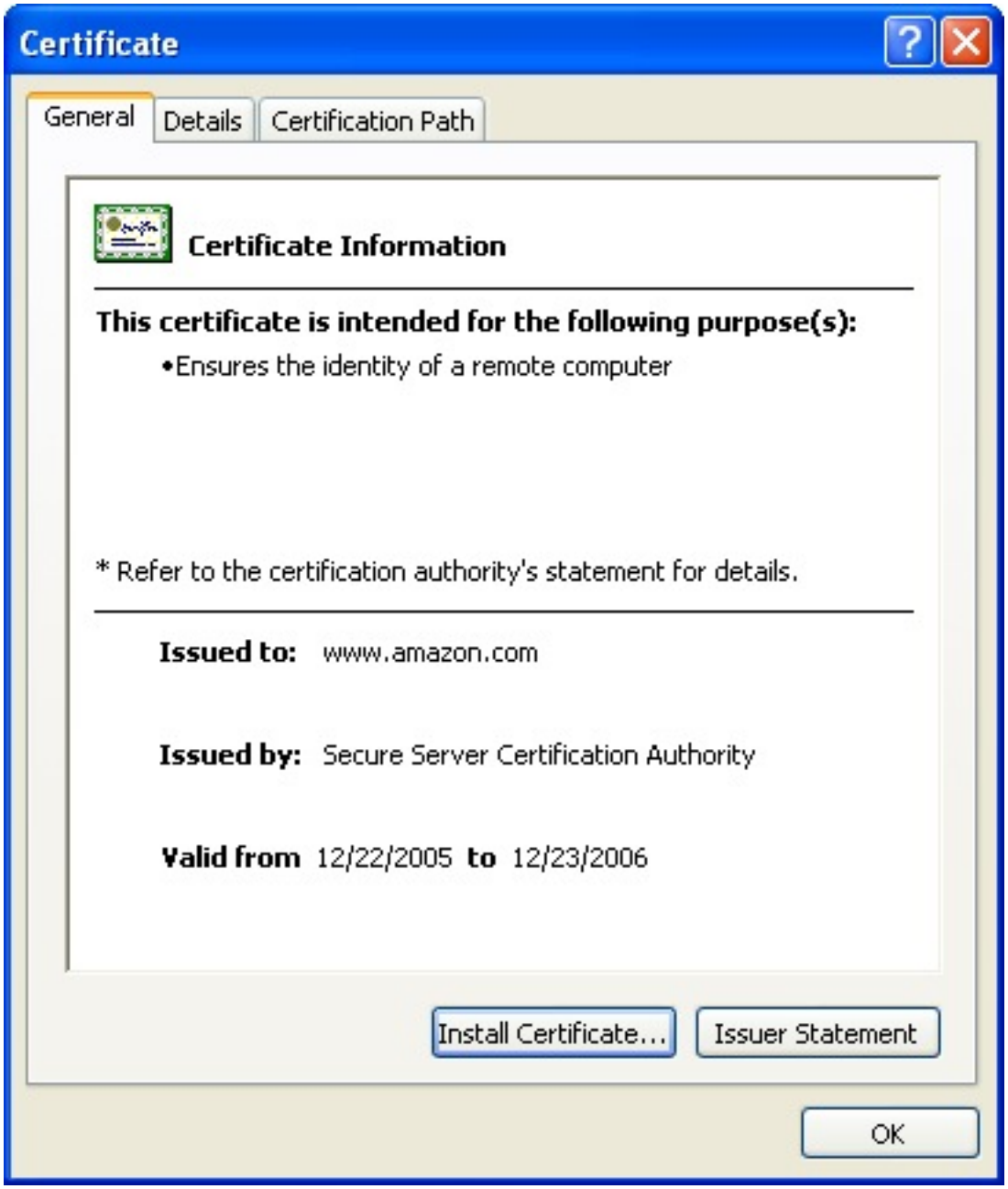

Figure 3. Web (SSL) certificate general tab

The certificate contains a good bit of information. To me, the three most important items it shows are:

1. Who is the entity?
2. Who vouches for that entity's identity?
3. What is the entity's encryption code?

Who is it?

Certificates are issued for various purposes. When a website uses a certificate for secure HTTPS, the *issued to* name (**www.amazon.com** in Figure 3) should exactly match the first portion of what you've typed into your browser's address bar. If it doesn't, most browsers will warn you that the certificate is suspicious.

But the focus of this article is on signing code, not on web shopping carts. (I haven't forgotten. Really!) For the purposes of purchasing a code-signing certificate, the *issued to* field can be pretty much anything that you can prove to the issuer – your own name or a company name. When I bought my certificate, I supplied a copy of the business license I have in my company's name.

Who vouches for that?

When it comes to vouching for a company's web site or application, it's all a matter of trust in the organization doing the vouching. Over the years, a number of Certification Authorities (CAs) have been established. As browsers and operating systems are updated, a current listing of those trusted authorities is supplied. You can see the root authorities your browser recognizes. If you're using Internet Explorer 6.0, click `Tools`, then `Internet Options`, then `Content`. Click the `Certificates` button. Then click the `Trusted Root Certification Authorities` tab. You'll see a list of your browser's trusted CAs.

You'll also see an `Import` button. Yes, this means you can tell your browser about other people/entities you want to trust.

You can get tools online to make your own digital certificates. If you have access to a copy of Windows 2000 Server (or 2003), you can install the Certificate Server service on it and create all the certificates you want for free.

But there's a slight problem here. Unless I can convince everybody I may encounter to consider *Jane's Fine Certificate Authority* as a trusted source, anything that uses the certificates I've created myself will still raise warning flags.

For a large enterprise, you can use Active Directory to instruct all the computers on your network to trust *Jane's Fine Certificate Authority*. But for an environment over which you don't have such control, you'll need help from a recognized CA.

Certificates are hierarchical. If I have a certificate issued by a trusted CA, then I'm trusted

as well. Or if I have my own corporation and my own certificate server, and on my certificate server I install a certificate from a recognized CA, then the certificates from my server will inherit that trustworthiness. Figure 4 shows a hierarchy containing one intermediate certificate service. I could click on either the top authority or intermediate authority to view its certificate details.

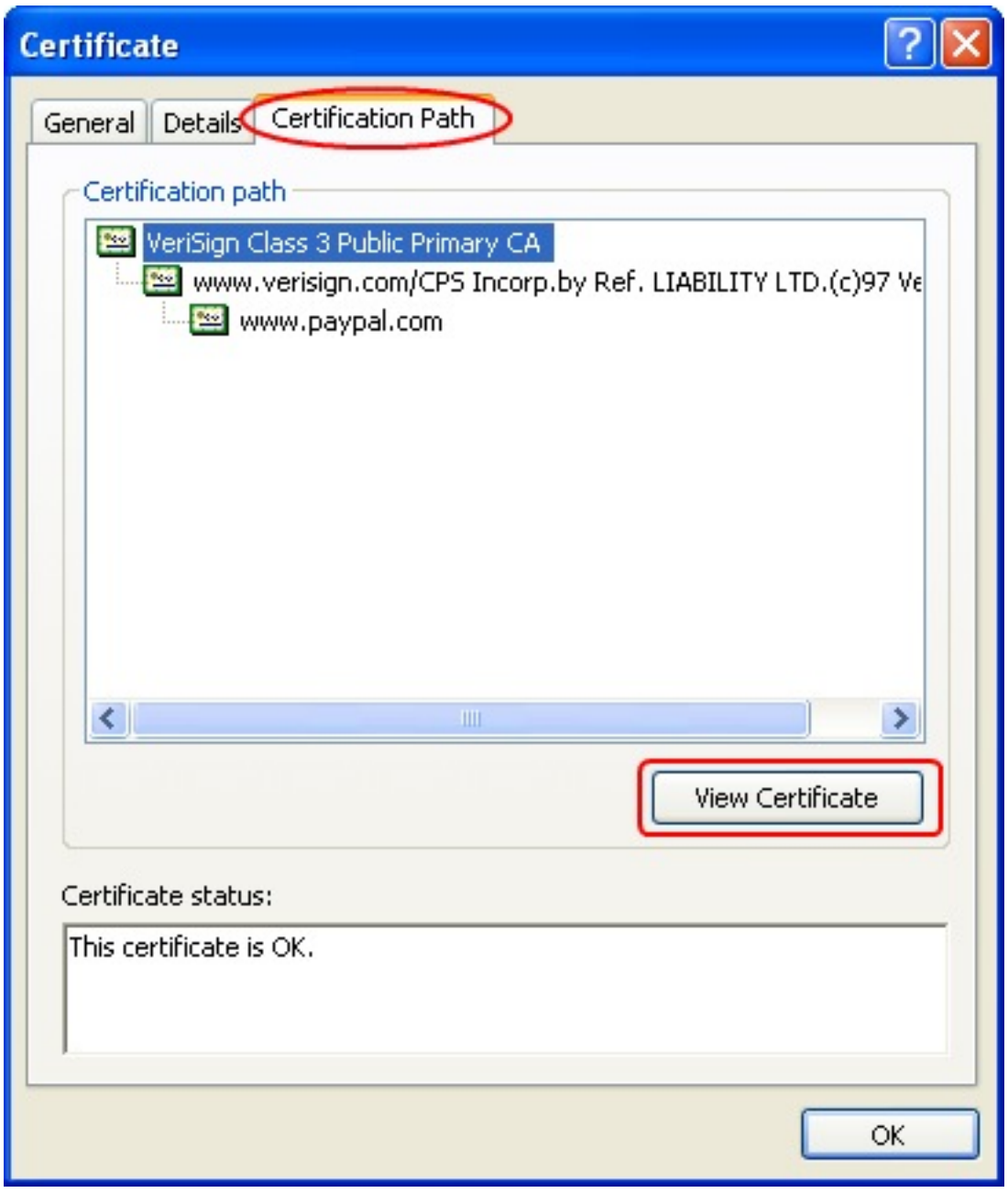

Figure 4. Certification hierarchy

What is the entity's encryption code?

Ah yes. There must be some trick to this, or anybody could make a certificate and call

himself Accounts-Receivable for Microsoft. The answer is encryption, and before I talk about how a certificate is encrypted I need to explain two types of encryption.

Symmetric encryption

I imagine most programmers have had at least a little contact with encryption at some point in their lives. The simplest encryption systems – and those most easy to understand – are *symmetric*. This means that the same password or secret phrase is used both to encrypt and to decrypt the material one is protecting. Here's an example:

You may have used the exclusive-OR (XOR) function at times to encrypt and decrypt simple strings in your software.

Exclusive-OR is a logical function that combines two numbers. If a specific bit of *both* the data byte and the encrypting byte is a 1 or a 0 (i.e., both bits are the same), the corresponding bit of the resulting byte will be a 0.

If a specific bit of the data byte is a 1 and the corresponding bit of the encrypting byte is a 0 (or vice versa), the corresponding bit of the resulting byte will be a 1.

In other words, the resulting bit will be 1 only if the two bits being combined are *not* the same.

In a truth table format, it would be expressed:

```
0 XOR 0 = 0
0 XOR 1 = 1
1 XOR 0 = 1
1 XOR 1 = 0
```

Let's say the data byte is the letter Q, which is an ASCII 81, and you're going to use a value of 182 to encrypt it:

```
81 =    01010001
182=   10110110
XOR= 11100111     (decimal 231
```

The encoded byte that this process has produced from the original letter Q is an ASCII 231. (Notice that either two ones or two zeroes have 'added together' to result in 0, but

where the bits don't match the resulting bit is a 1.)

Now comes the interesting part. When you XOR the result (231) with the encoding byte (182), you wind up with the original ASCII 81 (the letter Q). So you've used the same key (in this case, just a byte with a decimal value of 182) first to encrypt the data and then later to decrypt it.

Here's a snippet of Clarion code that uses the exclusive-OR function to encrypt a string, using a key string rather than a single encryption value:

```
InputString   string(30)     ! string to be encrypted
OutputString  string(30)     ! encrypted output string
KeyString  string(30)        ! my secret key
I   SHORT

  CODE
  KeyString='vsdfasdf*(*)@$ljasdbe5qwe'|
     & 'Pw2rqasdvvQWEQRWEVqdva'
  loop i=1 to len(clip(InputString))
    OutputString[i]=chr(bxor(val(InputString[i]),|
      val(KeyString[i])))
  end ! loop
```

You could also dimension a byte OVER the string to avoid having to flip back and forth between VAL and CHR. But don't use a CSTRING or it will see a byte that's wound up set to zero as a terminating character. For longer strings, you can loop back through the key string.

To decrypt OutputString, just repeat the exclusive-OR process.

This demonstration used a simple exclusive-OR, but there are many other algorithms.

Upside and Downside of Symmetric Encryption

The upside to symmetric encryption is that it tends to be fast and is easy to understand.

The downside is that you need to keep KeyString a secret, *but* you need to share it with anybody who needs to be able either to encrypt or decrypt your material. The wider a secret is shared, the less secure it will be.

Asymmetric Encryption

The solution to not having to share a secret encryption key is to use asymmetric encryption.

Here, instead of using a single key both to encrypt and to decrypt the material, you use one key, called a *public key*, to encrypt, and *the other* key, the *private key*, to decrypt. You must keep your private key secret, but because information can be encrypted with your public key you can still manage to exchange information with other people.

Let's go back to the list of certificates in the web browser. On the `Details` tab (in IE 6.0 – other browsers may have a different location for the data) one of the items of information available to the user (or to a browser or other program) is the public key (Figure 5).

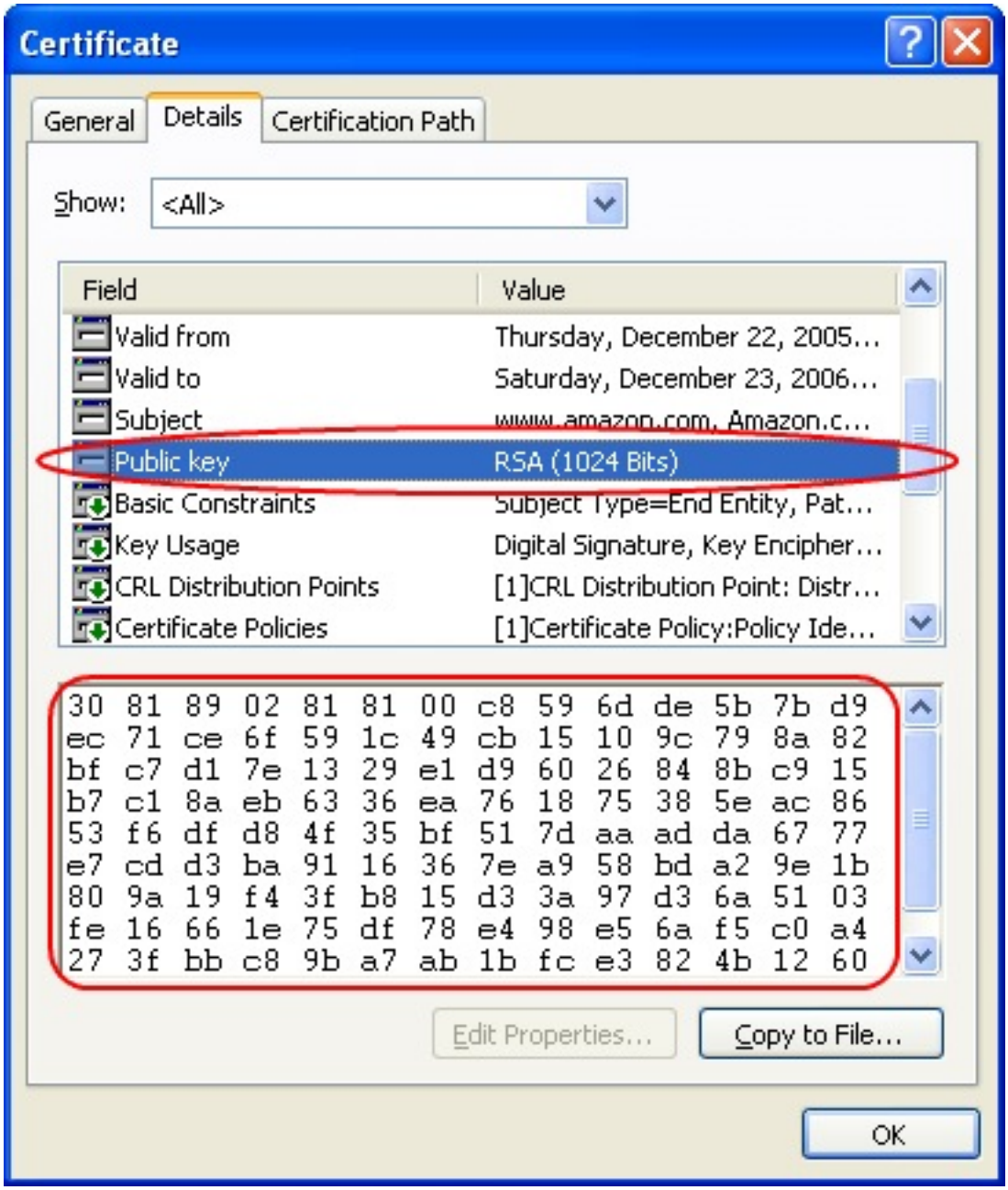

Figure 5. Public key

There are two ways to use public/private keys: encryption, and signing.

Encryption

If I give you my public key, you can encrypt information and send it to me. Nobody else can read it unless they have access to my private (secret) key. In fact, I can give my public key to everybody. (Hey, that's why it's called *public*!)

What if I need to send information back to you? It can't be done securely with my key

system. But you can send me your public key, and I can use it to encrypt my replies to you. Then you'd use your private (secret) key to decode what I sent. (A much more elegant method, SSL (Secure Sockets Layer), combines public/private keys and symmetric keys; I'll describe that briefly a little later.)

Signing

Aha, signing! Something that's beginning to sound relevant!

If I use my private (secret) key to encode something, anybody can use my public key to decode it. That's not very useful for keeping information secret. But if I use a strong encryption algorithm, the very fact that my public key decodes the information is a pretty sure sign that my private key encoded it (or 'signed' it). Using the private key to sign something is a guarantee of its validity (sometimes referred to as non-repudiation). Note that to sign a piece of software, I don't need to encrypt the entire file. The signing process creates a hash of the file, which is then encrypted. If a hash created on the user's computer doesn't match the hash the author encrypted in the file, then the file has been altered.

The browser or operating system that opens your software will use the public key information to be sure you signed it, and will use its copy of the Certification Authority's public key. If your information is traceable back to an authority that the browser or operating system trusts, the user will see something like Figure 6, instead of Figure 1.

Figure 6. The same program as Figure 1, but now it has been signed

If I click on the link next to the `Name:` prompt, I'll be taken to Beach Bunny Software's website.

If I click on the link next to the `Publisher:` prompt, I'll see the digital signature information (Figure 7).

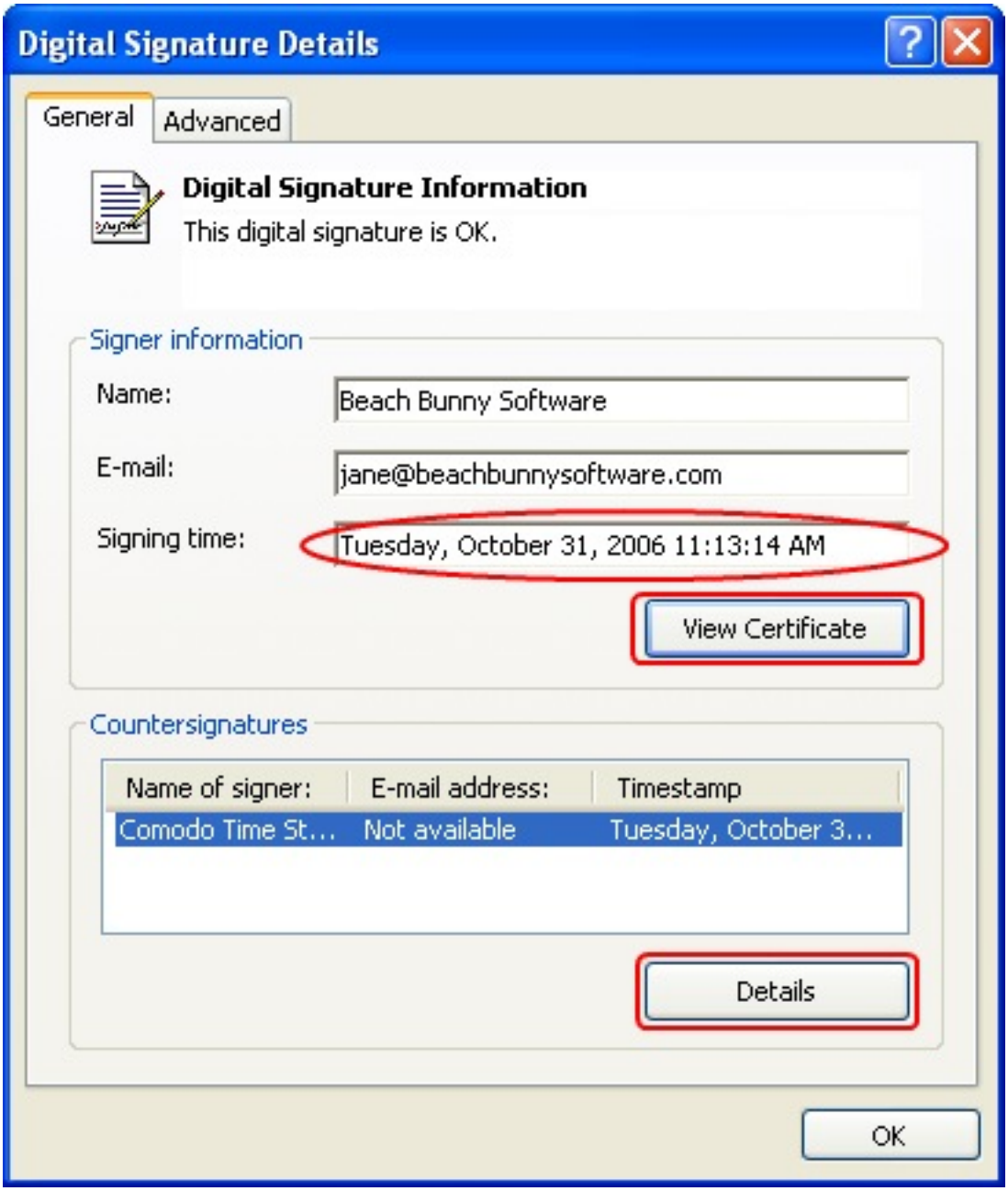

Figure 7. Signature details

In Figure 7, note that the date and time at which the program was signed are specified. The certificate I purchased is valid for two years. Because the date and time of signing are specified, even after my certificate expires, this program won't set off warning bells because it was signed while the certificate was valid. The signer in the

`Countersignatures` section is the time stamp service that verifies when the program signing was performed. (For this to be done, I needed an Internet connection at the time I signed the program - more on that later.) Clicking the `Details` button will bring up the certificate of the time signing server. Clicking `View Certificate` will show details of my signing certificate (Figure 8)

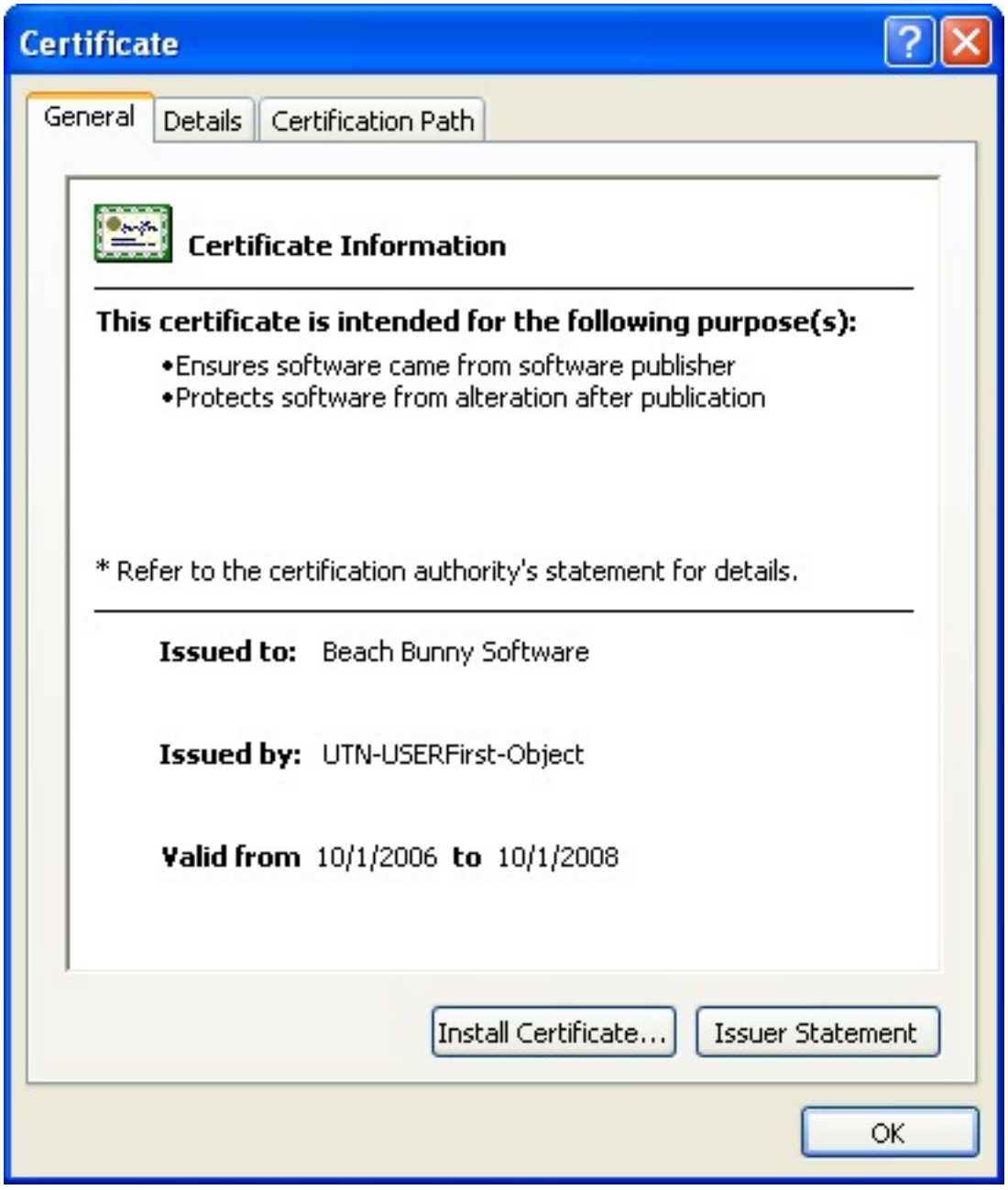

Figure 8. Details of my code signing certificate

Notice that my certificate in Figure 8 is similar to Amazon's (Figure 3). But there are differences. There are a number of types of digital certificates. A certificate for code signing will not work as an SSL certificate on a web server. An SSL certificate won't work to authenticate a certificate server. When you purchase a certificate, be sure you buy the

type you need.

Combining Public/Private Keys with Symmetric Keys

You may have noticed that Amazon's public key (Figure 5) is a 1024-bit key. But what about the 128-bit SSL encryption our web browsers are so proud of?

SSL (Secure Sockets Layer) is the protocol used for secure web page (HTTPS) exchanges. Glossing over details (Microsoft has a good description of those in the SSL handshake reference linked to at the end of this article), when I go to check out at Amazon.com over an HTTPS connection, my browser first gets Amazon's public key. It also checks the certificate to make sure it's really talking to Amazon and not to Joe Hacker. My browser invents a secret 'master' key for this session. It uses Amazon's public key to encrypt the master key, and sends it to Amazon. Assuming only Amazon has access to Amazon's private key, only Amazon can read the secret master key my browser has created. After some more negotiation, both Amazon and my browser create one or more *symmetric* session keys based on that master key my browser sent at the beginning of the exchange, and they start using them to encrypt the subsequent flow of information back and forth. With modern browsers, these session keys are 128 bits in length. As mentioned earlier, symmetric keys have much less computational overhead and can be processed quicker than public/private key systems.

For that matter, Microsoft's Encrypting File System also uses a combination of key types. When you encrypt a file, the system invents one or more secret symmetric session keys for that file. It then uses your public key to encrypt a copy of the session key(s), and stores the copy in the file's header. When you go to use the file the next time, it uses your private key to decrypt the session key(s), and then uses the session key(s) to decrypt the file.

The file's header also stores at least one more instance of the encrypted session key(s). The *Recovery Agent* public key is also used to encrypt a copy of the session key(s). This means that if a user's account is deleted or his profile becomes corrupt, the Recovery Agent should be able to recover his data. In an Active Directory domain, the Recovery Agent is typically the first Administrator account on the first domain controller (*not* the Administrators group). On a standalone computer, the Recovery Agent is the original Administrator account. This is why it is *not* a good idea to use EFS on a standalone computer when you're signed on as the Administrator. Eggs and baskets come to mind…

Next time

At this point you should have a usable grasp of basic encryption and signing concepts. In Part Two I'll show how to actually sign your applications.

Jane Fleming is a college dropout who subsequently lived four years in Europe, a year and a half in Mexico, and three years in India, and later taught yoga for a living in California (she's been vegetarian since 1970). She developed circuits and wrote assembly code for several embedded microcontroller projects during the 1980s. She began using Clarion Professional Developer for in-house projects back when Clarion was running display ads in InfoWorld and has used it very intermittently since. She is a former Microsoft Certified Trainer and taught Microsoft and Novell network administration at a business college for four years. Now widowed ten years, Jane plays classical piano and has found her métier as a semi-retired NRA-certified pistol instructor.

Reader Comments

Add a comment

- » Thanks for the very nice overview.
- » Now, I understand some things. Thank you.
- » Thanks for the great explanation.

Clarion Magazine

Clarion 101: Describing your Data

by Bruce Johnson

Published 2006-10-27

As you may have gathered by now, Clarion was originally designed as a tool for making business applications. It was made by, and then used by, people who did not necessarily have a computer background. Most Clarion DOS users were trained in some other field, and came to Clarion because it provided an easy way for them to write programs suitable for their fields. One of the reasons for Clarion's popularity is the way it handles data.

All programs can be broken down into three phases.

- Data input
- Data manipulation
- Data output.

Clarion for DOS was so successful because it recognized that most languages specialized in data manipulation, and data input and output were largely forgotten. Contrary to the fashion at the time Clarion shipped with excellent tools for building input screens and output reports, and also had an excellent interface for reading and writing data.

As Clarion matured it kept the ability to read and write data (more on that in a separate article) but it also improved the manner in which the data was described, by the programmer, to the program.

Normalizing your Data

At this point it's worth stopping for a moment and considering the way data is organized. For Computer Science graduates this is simple, and for many others they've learned it

along the way, but for the sake of all those without a classical education let's recap.

Firstly it's important to visualize the data as a list. The list is divided into rows, and columns. (Think Excel.) Each row contains one record. Each column contains a bit of information that belongs to that record.

For example:

Bruce	Johnson	Male	Married
David	Harms	Male	Married
Robert	Zaunere	Male	Married

In the above table each row (called a *record*) belongs to 1 person, and each column contains a different part of the information about that person. All the columns (known as *fields*), contain the same part of the information, for example the first field is the person's First Name.

This data can then be stored on the disk, where it's known as either a *table*, or a *file*. Clarion started off by calling these files, because each table was mapped to a physical file on the disk. Outside of Clarion people refer to them as tables, so along the way Clarion made an effort to call them tables as well. However some parts of the documentation, and some commands, still refer to them as files.

Notice that the above table contains a fair amount of repetition. For example all the people are male, and married. Storing these same words over and over again is inefficient. What if the system used 1 for Male and 2 for Female? What if numbers were assigned to the various marital statuses? Then there could be three tables that look like this:

Person

Bruce	Johnson	1	2
David	Harms	1	2
Robert	Zaunere	1	2

Sex

1	Male
2	Female

Status

1	Single
2	Married
3	Divorced
4	Widowed

You'll notice that each table has a name, so they can be told apart.

The above arrangement has a number of advantages.

1. The tables are compact. In other words each row in the Person table requires less physical space, both on disk and in memory. This isn't a big deal with three people, matters somewhat if you have three thousand people, and matters more if you have three million people.
2. The data can be easily changed. For example the word "Married" can be easily translated, or if misspelled it can be corrected in a single row in a single table. All the information is only stored once.
3. The data can be entered (by hand) faster. Instead of having to type out "Married" (7 characters) the user can type "2" (1 character).

The process of splitting up information like this is called *normalizing* the tables. This is something you do at design time, before you even start entering your tables into the Clarion dictionary.

Consider another example. Let's take the case where someone has walked into a store and is purchasing three items, a hammer, a screwdriver and a saw. The owner of the business needs to make an invoice, which will contain each item sold. If only one table is used then the table would look like this:

Name	**Address**	**Date**	**Item**	**Price**	**Quantity**
Bruce	Flat 4b	Dec 1 2007	Hammer	12.99	1
Bruce	Flat 4b	Dec 1 2007	Screwdriver	4.99	1
Bruce	Flat 4b	Dec 1 2007	Saw	35.62	1

What happens if Bruce returns 100 times to the store? And then changes his address? How many records would need to change? Assuming he's bought 300 items, 300 records would need to change. That would be a very inefficient task.

By the same logic, you'll notice that the date is being repeated three times. Since this one sale took place on one day that is to be expected. But there might be other information captured with this invoice that is also repeated three times, such as the name of the salesman. What about the product description? If the store sells ten hammers today should

the program store the word Hammer ten times?

To normalize this data you need four tables, one for the customer information, one for the product information, one for the invoice, and another for each line item in the invoice.

Customers

CustomerId	Customer number	Name	Address
1	1	Bruce	Flat 4b

Products

Product Id	Product Number	Description	Price
1	1	Hammer	12.99
2	2	Screwdriver	4.99
3	3	Saw	39.99

Invoices

Invoice Id	Invoice Number	Customer Id	Date
1	1	1	Dec 1 2007

Line Items

Line Item Id	Invoice Id	Product Id	Price	Quantity
1	1	1	12.99	1
2	1	2	4.99	1
3	1	3	35.62	1

Real world tips

Notice that the Line Items table includes a price column, and so does the Product table. Since you know the product you know the price, and hence you don't need to store it in the Line Items, right? Not really. In the real world things often have one sticker price, but sell for another price. It's important at the design stage to tell the difference between data that is the same, and data that simply looks like it's the same.

There is one last item to think about when you are normalizing your data. In the above example you'll notice that all the tables have an Id column. This column contains a unique number for each row in the table. At first glance this appears as overkill. The Invoice

Number could function as the Invoice Id, the Product Number could function as the Product Id and so on. However, just like with pricing, the real world kicks in and good design now will save you a lot of pain later.

In the real world the users of your program will want to control the unique numbers that you are arbitrarily assigning to your tables. They will probably have numbers already allocated to their existing customers and products. Since they will see, and possibly even enter, the customer number they will insist on it being a certain value. Whether that value fits, or doesn't fit into your careful scheme is of no concern to them.

Worse they will have an annoying habit of wanting to change numbers that they can see, that are supposed to be fixed to a specific value. They'll happily tell you during the design phase that "Employee numbers cannot be changed" but forget to point out that employees who are made redundant, and then later on re-hired, can in fact get a new employee number. Read Tom Ruby's True Confessions: A Tale of Two Users for an illustration of the dangers of using user-visible fields to link records.

By having "hidden" Id fields that the customer can't ever see, you can use the values safely without having to worry about their desires. You use these hidden numbers to reliably connect data in one table with data in another table (or tables)

In summary

- Data is arranged into *tables*, which contain *records* (rows) and *fields* (columns).
- *Normalizing* the tables is the process of breaking the data into multiple tables, so that all the information is only stored in one place.

Recommended reading

- Tom Ruby's series on normalizing databases: Five Rules for Managing Complexity

---

Living in Cape Town, South Africa, Bruce Johnson is a part-owner of CapeSoft and has been programming in Clarion since 1992. He authored the successful "Programming in

Clarion's ABC" book and has been involved in some of Clarion's most popular accessories. When not programming he enjoys cooking, and sports - the one as a direct result of the other.

Reader Comments

Add a comment

Clarion Magazine

Recursive Adds

by Steven Parker

Published 2006-10-26

“Heads down data entry” makes the ability to add multiple records without having to return to a browse desirable. Even customers in lighter duty situations want it.

In the last millennium (a.k.a.“before ABC”), recursive inserts had to be done by hand (formerly known as the “tear your shirt off, beat your chest and roll your own” approach):

Listing 1. Classic recursive adds

```
Loop
  Get(Customer,0)
  Clear(CUS:Record)
  GlobalRequest = InsertRecord
  UpdateCustomer
  If GlobalResponse = RequestCancelled then Break.
End
```

Code like this is still useful, by the way. If you want to call an update procedure without first calling a browse, this is just what you would do (if you only want a single record at a time inserted, remove the loop and the `GlobalResponse` check). Not bad for a piece of code originally written for CDD.

If, however, record priming is necessary (because of autonumbered keys or default values in the dictionary), further steps are needed to use this code. Why? Field priming occurs in the browse. C4/ABC made this change, moving field priming out of the form and into the browse, to support Edit In Place. So, the form no longer knows about its default values (as

I recall, autonumbering, in the absence of priming, will occur when the record is saved; but other initial field values will not be primed).

If field priming is necessary, it is possible to call the `PrimeRecord` method in the data entry form. (For a complete primer on the way forms are called in Clarion and an explanation of priming in the form, see Calling Form Procedures. While written in the last century, it is still relevant and the techniques described still work. How To Ignore The Form Template discusses field priming further as well as dynamically enabling and disabling buttons.)

To avoid having to do record priming in the form (suppose you're reusing a form also called from a browse and you're worried about double initializing), the code in Listing 1 can be modified as follows:

Listing 2. Recursive adds á la ABC

```
Loop
  Get(Employees,0)
  Clear(EMP:Record)
  GlobalRequest = InsertRecord
  Access:Employees.PrimeRecord
  Access:Employees.PrimeAutoInc
  UpdateEmployees3
  If GlobalResponse = RequestCancelled then Break.
End
```

In the demo app accompanying this article, select `Browse | Browseless` from the main menu. The code is in the embed for this menu option. The only caveat is that the form cannot be MDI (you can't call an MDI window directly from an app frame) If you do not want to create a new, non-MDI window, consider `START`ing a Source procedure; this Source procedure would contain the code in Listing 2 and would allow reuse of an existing MDI update form.

For the highest speed data entry, when data entry operators really know what they are doing, browseless recursive adds are the way to go. For environments that are not data entry intensive, there are other options.

The default option

Clarion has provided a template prompt and supporting code to enable recursive inserts since Clarion 4.

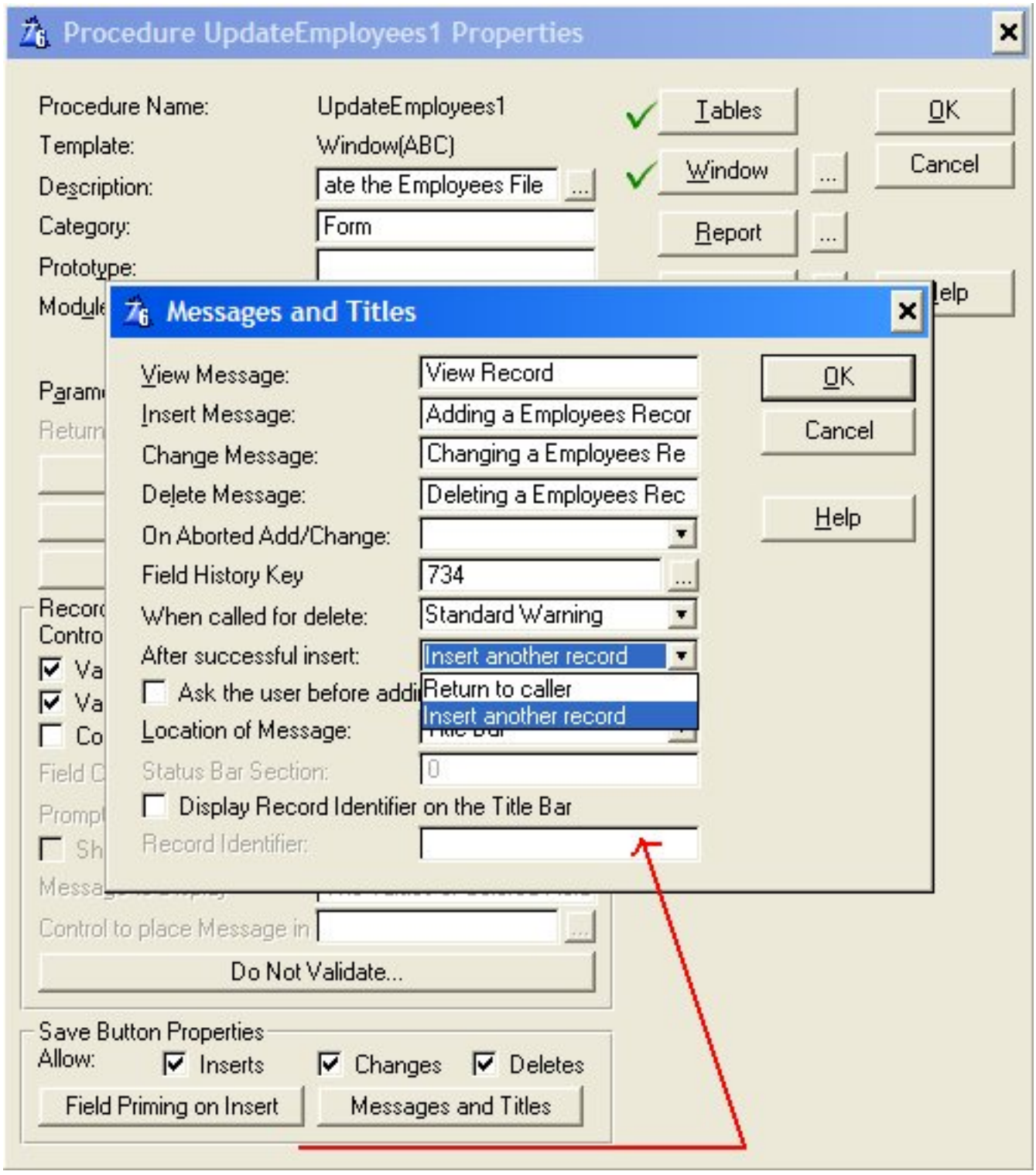

Figure 1. Template prompt for Recursive Adds

Simply by selecting `Insert another record` from the drop down, the data entry form automatically recurses. There is even the option to ask first (see the checkbox just below the `After successful insert` prompt).

This seems fairly flexible (especially considering the cost). Yet many users are still not happy. If I create a form and select `Insert another record`, the user has to press the `Cancel` button to end data entry. This, some users think, is not only unreasonable but constitutes unnecessary "work." As if such complaints were not enough, I have users who enter a new record and, seeing the form blank itself, either do not know what to do next or think the record wasn't created. (Neither training nor tool tips nor … help.)

Leave it to users to take a perfectly easy development task and turn it into a chore - as if their paying me somehow entitles them to make demands!

Buttons on the browse

On consideration, I realize that there are cases when a user might reasonably be expected not to want recursive inserts. For example, when creating a (sizeable) purchase order, recursive inserts are just what is wanted. But, a user editing an exiting purchase order might want to add just one item. In this case, automatically being in a recursive insert could be inconvenient.

With that in mind, explicitly making both data entry methods – single and batch -- available makes some sense (and the user is making a choice for which they are responsible). To affect this, I need to show my user both a standard insert button and a batch mode button:

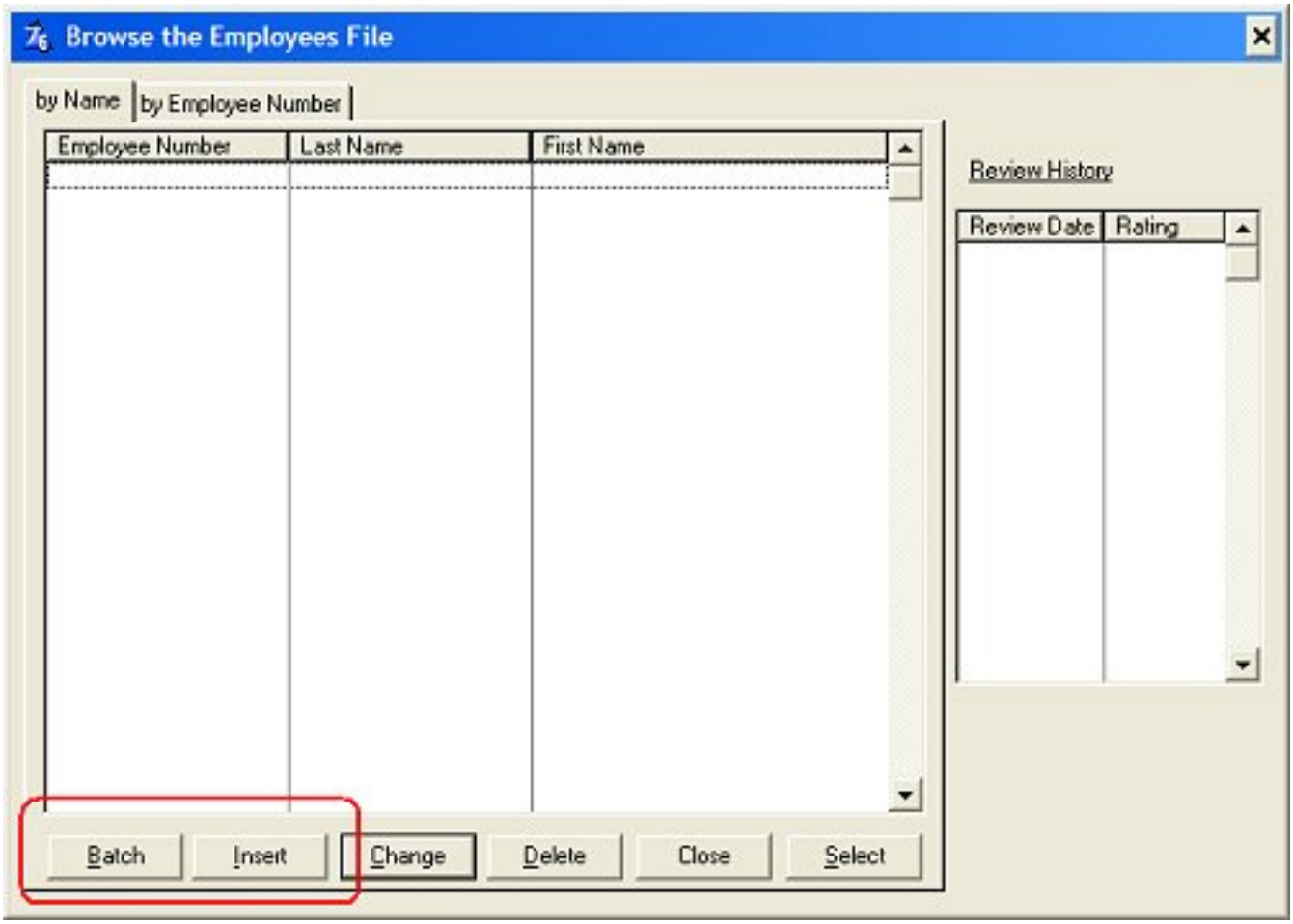

Figure 2. Single and Batch Insert buttons (view full size image)

To implement the two button approach, I could use the code in Listing 2 for batch mode and be done with it. The standard Insert button would handle single item inserts. Indeed, I could do this and the form would close and the browse flash on each iteration. As Henry Plotkin observed (Aesthetically Pleasing Recursive Updates), this looks very unprofessional.

Since the standard template implementation is quite smooth, it would be nice if I could piggyback what I want to do on what is already in the box. So, a quick look under the covers is in order.

Under the bonnet

I know that the form template supports recursive inserts. What I need to know is “How?” Examining the generated code, I discover that the templates use the `InsertAction` variable. (Here's how I find out what properties are being affected: open the form template – from the Clarion main menu, `Setup | Template Registry` – and search for the

prompt text. Then trace the template symbol to something that looks like code.)

If I look at the code generated when `Return to caller` is selected, I see this code:

```
IF SELF.Request = ViewRecord
  SELF.InsertAction = Insert:None
  SELF.DeleteAction = Delete:None
  SELF.ChangeAction = 0
  SELF.CancelAction = Cancel:Cancel
  SELF.OkControl = 0
ELSE
  SELF.CancelAction = Cancel:Cancel
  SELF.OkControl = ?OK
  IF SELF.PrimeUpdate() THEN RETURN Level:Notify.
END
```

If I now select "Insert another record," I see the same code, with one addition (in bold):

```
IF SELF.Request = ViewRecord
  SELF.InsertAction = Insert:None
  SELF.DeleteAction = Delete:None
  SELF.ChangeAction = 0
  SELF.CancelAction = Cancel:Cancel
  SELF.OkControl = 0
ELSE
  SELF.InsertAction = Insert:Batch
  SELF.CancelAction = Cancel:Cancel
  SELF.OkControl = ?OK
  IF SELF.PrimeUpdate() THEN RETURN Level:Notify.
END
```

This suggests that I can, at run time, change `SELF.InsertAction` from nothing (property not assigned, as in Figure 2) to `Insert:Batch` (as above; the other values `SELF.InsertAction` can take are`: Insert:None` – inserts not allowed; `Insert:Caller` – return immediately after inserting, this is the template default; and `Insert:Query`). Or, if I have designed the form to use `Insert:Batch`, I can change it to `Insert:Caller`. Specifically, it looks like I can pass in parameters, now that Clarion 6 supports full parameter passing, to the form and set `Self.InsertAction` on the fly.

Where? How? Design decision time.

And now, back to our show

In the demo app, `Browse | Single Add/Batch Add Buttons` calls the browse shown in Figure 2, above. If you press the `Insert` button and add a record, you are returned to the browse. If you press the `Batch` button, you are in recursive add mode and have to press `Cancel` to exit the form.

The procedure name is `BrowseEmployees` if you want to follow along.

While there are two buttons, there is only one update form. The “trick” is in passing a parameter to the form, a feature not fully supported in the stock templates until 6.x:

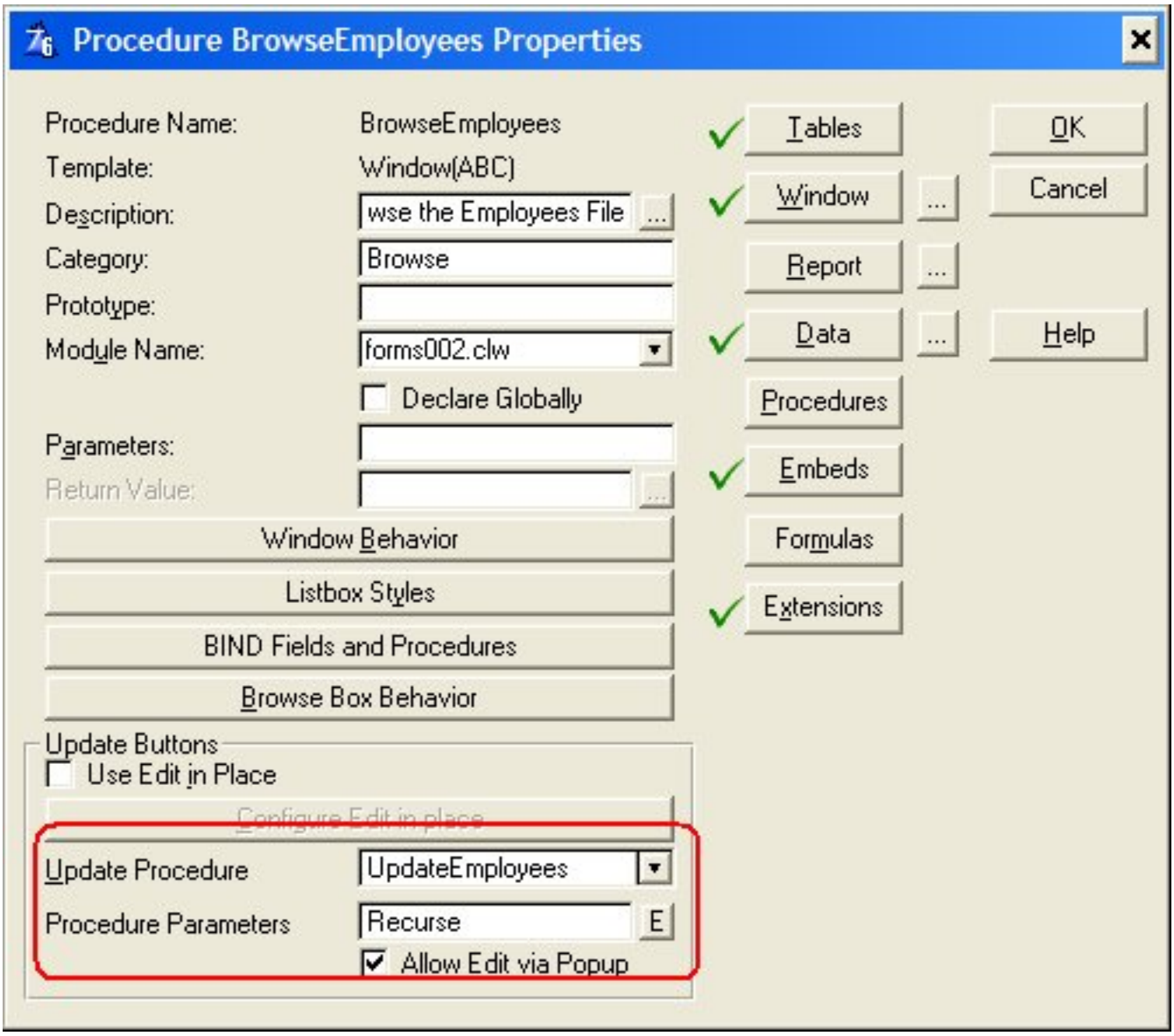

Figure 3. Passing a parameter to the update form

`Recurse` is a local variable and is set when pressing the `Batch` button (by default, the form will be single insert). In fact, the sum total of code in `BrowseEmployees` is in the `Batch` button's `Accepted` embed:

```
Recurse = 1
Post(Event:Accepted,?Insert:2)
```

The `Post` statement is how two buttons call only one form. For complete reliability, `Recurse` should be cleared, `ResetFromAsk` looks like a good place to do that (it is after the call to the update procedure).

Everything else happens in the update form, `UpdateEmployees`. Because the browse passed a parameter, the form has to be prototyped to receive it:

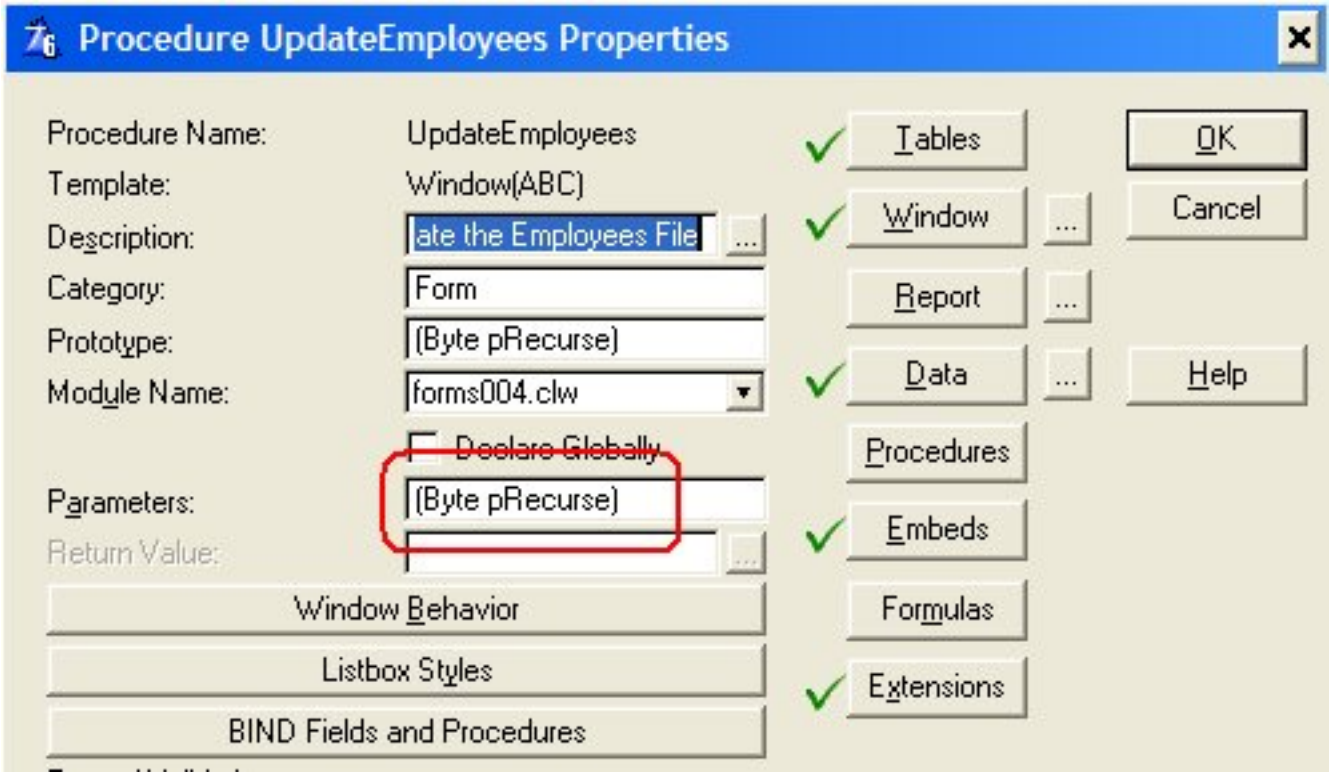

Figure 4. The form's matching prototype

The magic happens in `ThisWindow.INIT`, reading and using the parameter:

```
If pRecurse and Self.Request <> ViewRecord
  Self.InsertAction = Insert:Batch
End
```

This codelet need only be placed after the template generated code shown in Figure 2. In

the demo app, you will find it at priority 7600, just after `OpenFiles`.

And, there you go! It works; it's smooth.

And, sure enough, there are users who are unable to figure out in advance that they want to add several records. These users also cannot be bothered to finish one record, return to the browse and click the `Batch` button (and my boss wonders why I consider spec a moving target). These users want…

Buttons on the form

Buttons on the data entry form, as shown in Figure 5, allow users to decide, after beginning data entry, to go into batch mode. (Actually, I default this form to batch mode and use the button on the left to break out of batch mode.) Given how simple, relatively, buttons on the browse were to implement, I am optimistic about buttons on the form.

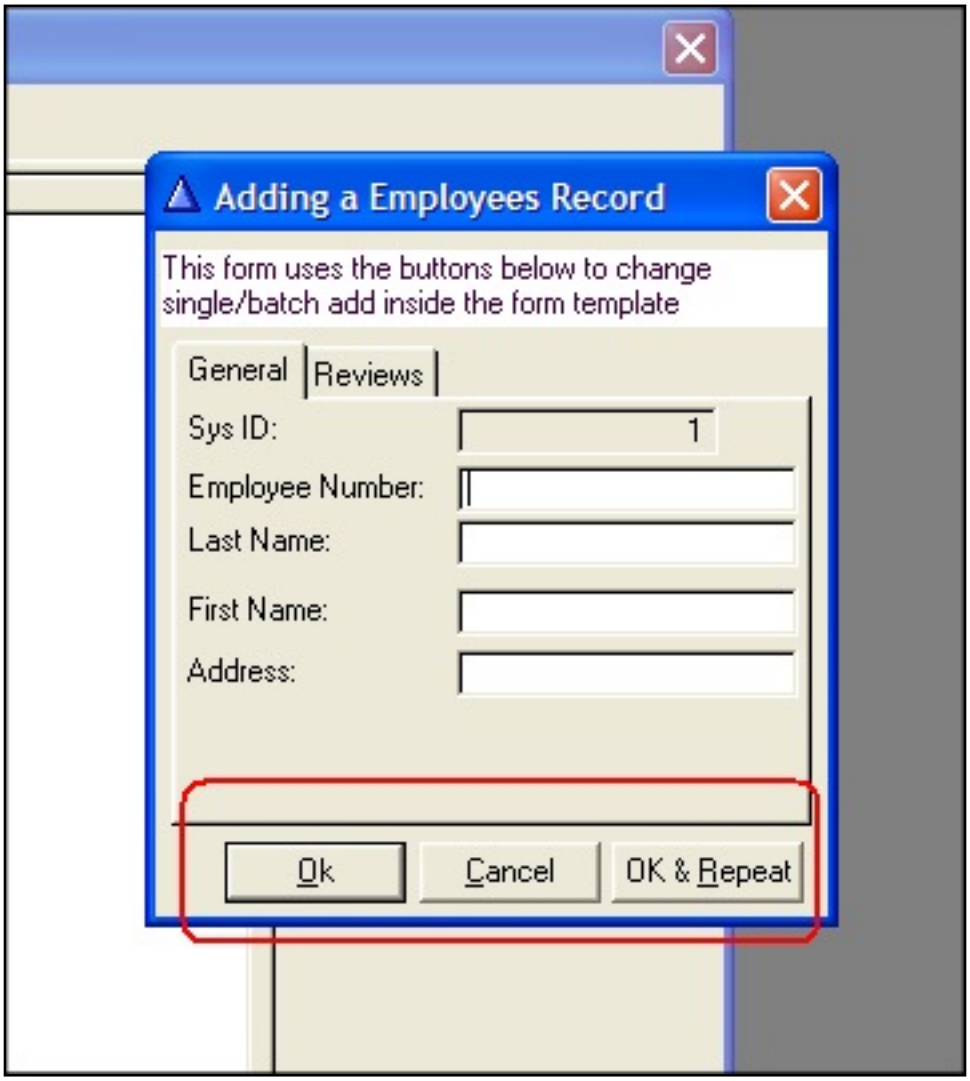

Figure 5. Buttons on the form

My concern is whether or not I can reset `Self.InsertAction` after the window is fully initialized. If I cannot, I'm hosed. If I can, Listing 3 supplies the answer.

Listing 3

```
Self.InsertAction = Insert:Caller
Post(Event:Accepted,?OK)
```

Inserting Listing 3 in a purpose-created button's `Accepted` embed will do. This assumes that I set the default behavior to be Insert another record. Thus, the template `Ok` button is renamed `OK & Repeat` and the button displaying `Ok` is actually a new button. If the form was created to start in recursive mode, the `Ok` button would be the template's `Ok` button and `Ok & Repeat` would be the purpose-created button, with the code in Listing 4.

Listing 4

```
Self.InsertAction = Insert:Batch
Post(Event:Accepted,?OK)
```

In the sample app, `Single / Batch Add From Form` demonstrates this technique. As it turns out, while `InsertAction` is set in `INIT`, it is used in several other places. The last of these places is `TakeCompleted`. So, if I change `Self.InsertAction` before `TakeCompleted`, I can bail out of whichever mode the form is in using the code in Listing 3 or 4 in the new button.

Eureka! Done.

Summary

There are four ways to do recursive inserts. Two ("roll your own" and the way built into the templates) are not dynamically changeable. Two (buttons on the browse, buttons on the form) are end user configurable.

More instructive, however, is my trick of using the template's own code to trace down the variables and methods necessary to accomplish a goal. By examining the form template, I

discovered `InsertAction` and that made changing entry modes at run time almost easy.

Download the source

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

Reader Comments

Add a comment

- » I like the recursive add to happen in the Browse so the...

Clarion Magazine

Classes For Background Processes

by Dave Harms

Published 2006-10-20

One of the most-anticipated features of Clarion 6 is also one of the least-used. It is the ability to run "worker" threads in the background. Worker threads, often called daemon threads in the *nix world, are processes that don't interact with the user. They may be used for retrieving emails, spooling print jobs or any of a number of tasks.

So threads are a great way to handle background tasks. But threads seem very procedural: you start some code running, and it runs until it ends. What if you want to use threads in an object-oriented environment? Is it even possible to have a class that runs on its own thread? How do you control threads in an object-oriented environment?

I'll try to answer some of those questions, but to make sure that everyone's clear on the background thread concept, I'll start with some procedural code.

A background procedure

The simplest way to create a background process is to `START` a procedure (one that doesn't open a window) and let it sit in a `LOOP`, punctuated by `YIELD` or `SLEEP` statements. Here's an example of a thread that makes a beep noise once per second:

```
GoBeep  procedure
  code
  loop
     sleep(1000)
     beep
  end
```

Sleep, by the way, is a straight call to the Windows API function by that name. You prototype it this way:

```
  map
      module('winapi')
          sleep(long),raw,pascal
      end
  end
```

All Sleep does is tell the operating system to ignore this thread for the specified number of milliseconds.

You can call the procedure this way:

```
START(GoBeep)
```

There's only one problem with the above code: once you start `GoBeep`, it'll never stop running. You can shut down your application, but that loop is still going to execute, and you'll have to kill the app off in the task manager. So obviously you need a

way to shut down the loop. The easiest way to do this is with a global variable:

```
GoBeep  procedure
  code
  loop
     sleep(2000)
     beep
     if glo:stopped
       break
     end
   end
```

All you need to do is set glo:stopped to a non-zero value, and the loop terminates. Easy, if not particularly neat.

You can test this code in the example application. Choose the Test menu, and then select Procedure Test. The background thread will begin and you'll hear the beeps. Then select Test|Disable Procedure Test to toggle the value of `Glo:Stopped`. A check mark will appear beside that menu item, and as long as the check mark shows (meaning `glo:Stopped = 1`) any running instance of the background procedure will immediately terminate.

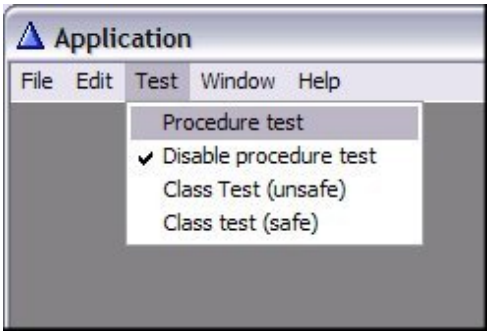

Figure 1. The Disable Procedure Test option toggled on

But what if you want multiple copies of this procedure running at the same time? Setting `Glo:Stopped` will shut them all down. Other mechanisms suggest themselves – you could pass in the address of the global variable to use as a shutdown mechanism. Or you could poll a global procedure for a shutdown notice. But all of this takes away from the desired unity of a worker thread. Ideally you want something just a bit more self-contained. And that leads to the idea of a class that can run in the background.

Almost STARTing a class

Actually, you can't run a class in the background. START only works with procedures, not classes. How about running a class's method in the background? As it turns out, you can't do that either. That's because the implicit first parameter of any method is the class itself (well, technically the class's instance, a.k.a. the object). START doesn't allow a class/object as a parameter.

START does, however, allow you to pass up to three strings to a STARTed procedure, and this turns out to be a very useful feature.

If you can't START a class, or a class's method, then the next best thing is to START a procedure, and pass the object's address to the procedure as a string. The procedure's job is to use the address to locate the object, then sit in a loop and call the object's methods as needed.

The example

Take a look at the following class declaration, contained in cciworkr.inc:

```
!ABCIncludeFile

 OMIT('_EndOfInclude_',_CCIWorkerPresent_)
_CCIWorkerPresent_ EQUATE(1)

cciWorkerClass
class(),TYPE,MODULE('CCIWorkr.CLW'),LINK('cciWorkr.CLW',_ABCLinkMode_),DLL(_ABCDllMode_)
Running                     byte(0),private
StopThread                  byte(0),private
Start                       procedure
Stop                        procedure
Stopped                     procedure,byte
DoTask                      procedure,private
                       END

  _EndOfInclude_
```

`cciWorkerClass` has two properties, `Running` and `StopThread`. `Running` is set to true as soon as the thread begins, and to false when the thread exits. `StopThread` is a flag that the background thread must check each time through the loop. You could just use one flag to indicate a stopped/started thread, but there may be a delay between the request to stop and the actual termination of the thread, depending on the nature of the task. It's safer to track the "stop requested" state separately from the "stopped" state.

The `Start` method `START`s a procedure (you'll see that declaration in a moment), and the `Stop` method sets the `StopThread` flag. The Stopped method is called by the `START`ed procedure to determine when to exit. And finally the `DoTask` method contains the code the background process is to call on a regular basis.

Here's the first block of source code from cciworkr.clw:

```
MEMBER

  include('cciworkr.inc'),once


    map
        WorkerThread(String addr)
        module('winapi')
            sleep(long),raw,pascal
        end
    end
```

This class source file has a local map, which contains two procedures used by this class. The first is the procedure that will be `START`ed; the second is the Windows API Sleep call.

The key here is that although cciworkr.clw contains the methods for the `cciWorkerClass`, it also contains the associated procedural code. Having everything in one location eases maintenance.

Here's the code for the `WorkerThread` procedure. This is absolutely straight procedural code – it isn't part of the class. But there is an important trick right after the code statement.

```
WorkerThread        procedure(String addr)
worker  &cciWorkerClass
    code
    worker &= (addr)
    if worker &= null
        message('worker object is null - exiting worker thread')
```

```
            return
        end
        worker.running = true
        worker.StopThread = false
        loop
          if worker.StopThread
            break
          end
          sleep(1000)
          worker.doTask()
        end
        worker.running = false
```

This procedure contains a local reference variable of type `cciWorkerClass`. That means it can point to any `cciWorkerClass` object. If you look ahead a little to the `Start` method you'll see that the first parameter to `WorkerThread` is the address of the current object. `WorkerThread` then does a reference assignment:

```
worker &= (addr)
```

The parentheses are essential – they convert the numeric `addr` string to an actual address. Without the parentheses you'll get an illegal reference assignment error.

Now that `WorkerThread` has an object to work with, it can call methods on that object. It also checks the `StopThread` property to know when it should break out of its loop and resets `StopThread` to `False`. Each time through the loop the procedure calls the `DoTask` method to do whatever useful work the class can do. In this case the task is, again, just to emit a `BEEP`.

Here are the class methods:

```
  INCLUDE('CCIWORKR.INC')

cciWorkerClass.DoTask                       procedure
    code
    beep


cciWorkerClass.Start                        procedure
    code
    if ~self.running
       self.Running = true
       start(WorkerThread,,address(self))
    end


cciWorkerClass.Stop                         procedure
    code
    self.StopThread = false


cciWorkerClass.Stopped                      procedure
    code
    return choose(self.running=0,true,false)
```

The really handy thing about this approach is that the code that uses this class simply creates an instance and calls methods. Your declaration might look like this:

```
worker cciWorkerClass
```

Here’s the code to start the class:

```
worker.start()
```

And here’s the code to stop it:

```
worker.stop()
```

That’s pretty easy - starting and stopping the thread is as easy as calling a couple of methods. You can easily add a pause feature to bypass the DoTask method. And you can even have multiple instances of the class running and they won’t step on each others toes. Although the `WorkerThread` procedure is called multiple times, each has its own associated `cciWorkerClass` instance.

You can test this code with the Test Class (unsafe) option in the example application. You'll see the window in Figure 2.

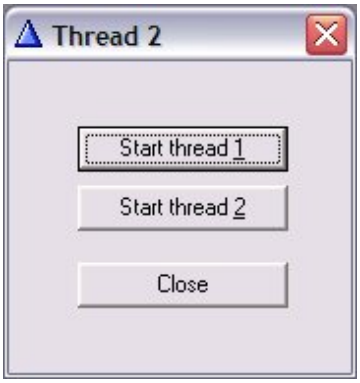

Figure 2. The test window

Click the two thread starting buttons, and you should hear two regular beeps in the background. Click the buttons again to turn off the threads.

But be careful - this option is labelled (unsafe) for a reason. If you start a thread, be sure you stop it before closing the window. If you don't close all background processes first, you'll get a GPF. That's because the thread is still running even after the procedure terminates, and is still attempting to access the `cciWorkerClass` instance which (as near as I can tell) has been trashed by Clarion's automatic garbage collection.

Constructors, destructors, and GPFs

You'll notice that I didn't use constructors and destructors in `cciWorkerClass`. In general, when running background processes, I prefer to exert control over when threads start up. This is especially important when you are dealing with multiple objects, where initialization must happen in a specific order, or when you need to set properties on an object before starting the thread (remember that you cannot pass parameters to a Clarion constructor).

It is, however, possible to use constructors and destructors with this background process technique, and in particular the destructor demonstrates an important technique for thread cleanup.

Here's a derived class which adds constructors and destructors:

```
cciSafeWorkerClass     class(cciWorkerClass),TYPE,MODULE('CCIWorkr.CLW')
Construct                   procedure
Destruct                    procedure
                       END
```

And here are the method declarations:

```
cciSafeWorkerClass.Construct        procedure
    code
    self.Start()


cciSafeWorkerClass.Destruct         procedure
    code
    self.Stop()
    loop
      if self.stopped() then break.
      yield
    end
```

In Clarion, `Construct` and `Destruct` are "magic" method names; these are the methods called when an application is allocated memory, and when it is disposed of. That's the case whether you declare a static object or use `NEW` and `DISPOSE`.

In the example application, choose the Class Test (safe) menu option. A window similar to Figure 2 will appear, but both threads will already be running. That means the beeps will be synchronized so you'll hear them as one beep, but after a few seconds they'll get out of sync (and it would be interesting to know why...). You can stop and start the threads with the buttons, but if you leave a thread running and close the window you will *not* see a GPF. The reason for that is the code in the `Destruct` method which requests thread terimination and then goes into a loop until the thread has actually terminated.

It's vital that the Destruct code loops until the thread has terminated; as long as the thread is running, terminating the object makes a GPF likely, if not inevitable.

Although this code works fine, I advise you do a lot of testing if you use an automatic destructor in a production environment. Since I'm not privy to the internals of Clarion's garbage collection mechanism, I can't guarantee that a class will never be trashed in this siutation. Perhaps the best approach is to explicitly call a cleanup method, and then have that same method called from the destructor, just in case your explicit call goes missing (and be sure that calling your cleanup method multiple times will not cause other errors or GPFs!).

Thread safety

You may have noticed that I'm not using a critical section or other mechanism to prevent simultaneous calls to the Stop or Start methods from mucking up the data. In part that's because these simple assignments are atomic; setting a `BYTE` or a `LONG` to a value happens all at once, and two threads can't contend for the data. But `USHORT` assignments are not atomic, and certainly `STRING` assignments aren't either.

The other reason I haven't used critical section is that there are only two threads involved; the one on which the object is created, and the one which the object itself creates. There is only one property which can be written by both threads at the same time, and that is `StopThread`. If the first thread somehow sets `StopThread` to a non-false value at an inopportune time, the worst thing that happens is the thread will terminate, and since the only way to set `StopThread` is to call `Stop`, I would assume that's what was supposed to happen. If the worker thread sets StopThread to false just before the loop, and just after a call to Stop by the first thread, then the thread will not stop, but that's not something that a critical section would fix anyway, and would have to be sorted out in a design review.

Summary

Although Clarion 6's true threading was hailed as a great step forward, worker threads remain underappreciated. Happily they're not that difficult to implement when they're wrapped up in an object-oriented framework.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Throwing Users Out: Methods of Computation

by Steven Parker

Published 2006-10-17

In the last action packed episode, I looked at ways of throwing users out of windows. While I did raise a sweat, I was able to do it without incurring any bodily damage (primarily to myself – defenestrating users can get quite messy). I did discover that there are more events being posted in some template procedures than I had, once upon a time, accounted for. But it all turned out right in the end.

One issue remains to be elucidated: how to figure out it is time to … dispose of the user.

There are three methods of computing this:

- The time of day method
- The countdown method
- The count up method

But first, the assumptions

I make one and only one assumption. I don't think it is an unreasonable one.

That assumption is that the amount of time a user is allowed to be inactive is stored in a variable. The variable can be in a configuration file, an INI file, a registry entry. Further, I assume that the variable is global. (This is a safe use of static data; it is read many times but very rarely, if ever, written.)

And now, another assumption

I make one and only one more assumption: my base unit is "seconds." So, either the inactivity variable is a number of seconds or is converted to seconds (see Marking Time, Part 1 for a full discussion of converting various expressions of time to other expressions and to Clarion time).

So,

```
Timeout = GLO:NumberOfSeconds
```

or

```
Timeout = GLO:MinutesAllowed * 60
```

to give me the seconds I expect.

Why seconds? I'm glad you asked.

I use seconds for two reasons. First, I can set a timer on a window for one second (100 in the Timer prompt on the Window Properties worksheet) and have matching units to work with all around. Using seconds means I do not have to do any further conversions.

Second, I believe that any timeout amount that cannot be comfortably expressed and understood in seconds is just too long.

Time of day

When thinking about "when do I give the user the boot," time of day is probably the first method that comes to mind.

When the user enters a procedure or when the user is active (does something), I can compute the time of day at which he/she should be thrown if they do nothing further.

To compute the time of day to kick them out I convert my number of seconds to Clarion ticks (okay, one more conversion *was* needed) and add to the current time:

```
Timeout = Clock() + (GLO:NumberSeconds * 100)
```

`Timeout` now contains the clock time at which to dump the user. `Timeout` is recomputed whenever there is a user event, so it is always current.

Then

```
If Event() = Event:Timer
  If Timeout => Clock()
    Message('Time to go!','Bye Bye',ICON:SoLong)
    ! but really Post(Event:CloseWindow)
    ! or Post(Event:Accepted,?Cancel)
  End
End
```

and the user is out.

This is all fairly straightforward. This is all very logical and, even, intuitive.

Except. Except that it embodies an assumption that can undermine the entire process: It does not allow for an application to be running at midnight. Midnight rollover is discussed in Marking Time 3: Inter-Date Computations and Replicating IDLE: All Quiet on the Keyboard? And the reader may want to take a moment to look at these articles.

So, while still fairly straightforward, midnight rollover handling can produce some hard to read code. One of my first demonstrations of handling this circumstance looked like this:

```
!TakeEvent
If Event() <> Event:Timer
  Timeout = Today() + |
    ((Clock() + (GLO:NumberSeconds * 100)) |
    / 8640000)
End
```

(so far, not too bad) and

```
!Timer
If Event() = Event:Timer
  If GLO:NumberSeconds > 0
    !new day, adjust for midnight rollover
    If (Today() > INT(Timeout)) |
```

```
        AND (Clock() + 8640000 > |
        ((Timeout - INT(Timeout)) * 8640000)) |
        OR (Today() = INT(Timeout)) AND |
        (Clock() > ((Timeout - INT(Timeout)) * 8640000))
         SELF.CancelAction = Cancel:Cancel
         Post(Event:Accepted,?Cancel)
    End
  End
```

(Ick.) This works by checking the dates first then making the appropriate time comparison. Alternately, I could create a second StarDate in the Timer embed and subtract:

```
  If Event() = Event:Timer
    If GLO:NumberSeconds
      StarDateNow = Today() + ( Clock() / 8640000 )
      If StarDateNow - Timeout > 0
        SELF.CancelAction = Cancel:Cancel
        Post(Event:Accepted,?Cancel)
      End
    End
  End
```

Effective, more readable than the previous codelet, but still much less readable than `If Timeout => Clock()`.

Countdown

I really shouldn't *need* to calculate the time of day at which to expel the user, at least not when I am willing to set my window timer to one second (for the reasons mentioned earlier). In other words, its intuitiveness notwithstanding, the time of day method is more complex than it needs to be, at least for general use.

Since I know the number of seconds (or minutes converted to seconds) after which to time out, I should be able to simply count down the remaining time. Something like:

```
  If Event() = Event:Timer
    Timeout -= 1
    If Timeout = 0
      !do something
    End
```

```
End
```

With a window timer set to 100 (one second), `Timeout` will be decremented precisely once per second. Isn't it just great when a plan comes together?

If I do this, then `TakeEvent` becomes a simple reset of `Timeout` to its initial value:

```
If Event() <> Event:Timer
  Timeout = GLO:NumberSeconds
End
```

Not only is this easier to read than setting the time of day or creating a StarDate, it is faster. Because it is a simple assignment, no computations, it executes just about as fast as code can execute.

More important, midnight rollover is automatically handled with nothing further to consider. (It might be more accurate to say that midnight rollover is simply ignored: time of day never even enters the equation at all). The sample apps accompanying this article (C55) implement this technique in `BrowseCustomers`, for example.

Counting down works. Even better, it allows displaying the countdown on the window (see `BrowseCustomers` and `UpdateCustomers1` in the sample app).

Counting down works with one small exception (is anyone here surprised?). Actually, on reflection, the exception isn't in the least "small."

If there is an MDI frame procedure and the "Display the date and/or time in the current window" extension template is used, the timeout will never be hit. This assignment:

```
Timeout = GLO:NumberSeconds
```

will be updated every time the date/time is updated in the frame. Placing this code:

```
0{Prop:Timer} = 0
```

in `ThisWindow.Init, Enter procedure scope (Priority 501)` will turn off the frame's timer. (Because this code is executed before the current procedure's window is opened, "0" still refers to the frame's window. The procedure's timer is entirely undisturbed.) In the sample app, I implement turning off the frame timer in `BrowseCustomers`. Try commenting out this line and you will see that the browse with

the "IDLE" code never times out. (`Prop:Timer` is restored at the very end of `ThisWindow.Kill`.) If you don't want to comment out and recompile, just open all the demo browses sequentially (not all of them touch the timer); `BrowseCustomers` will no longer time out.

There is a very important lesson in the previous paragraph: *code in one procedure can affect the window of another procedure*. If code is called before `Open(Window)` or after `Close(Window)`, the "current" procedure's window is *not* the one affected.

Count up

A count up timer is just a variation on a countdown timer (one can suppose that a developer's choice between a countdown and a count up timer says something about them psychologically but, happily, that is a subject for another time and place).

`TakeEvent` remains a simple reset of `Timeout`:

```
If Event() <> Event:Timer
  Timeout = 0
End
```

`Event:Timer` tests whether the maximum number of seconds has passed:

```
If Event() = Event:Timer
  Timeout += 1
  If Timeout => GLO:NumberSeconds
    SELF.CancelAction = Cancel:Cancel
    Post(Event:Accepted,?Cancel)
  End
End
```

Et voilá!

But wait! There's more!

Using any of the techniques discussed in this article and the last one allows two very interesting development techniques:

1) I can have any number of timers on any number of windows

2) I can have any number of inactivity checks on a single window.

The first is obvious. The second less so but, I think, rather important. Let's say I want to clear a particular field after `x` number of seconds and kick the user out after `y` number. These techniques make this as straightforward as creating the first codelet, copy, paste, and change variables names/actions.

```
If Event() <> Event:Timer
  Timeout = GLO:NumberSeconds
  ClearTime = GLO:ClearSeconds
End
```

And

```
If Event() = Event:Timer
  Timeout += 1
  If Timeout => GLO:NumberSeconds
    !do something
  End
  ClearTime += 1
  If ClearTime => GLO:ClearSeconds
    Clear(pre:Field)
  End
End
```

Now, *that's* pretty cool.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Clarion 101: Choosing a Database

by Dave Harms

Published 2006-10-12

In Parts 1 and 2 of this series I provided an overview of the Clarion IDE, and at the end of Part 2 I described the typical application workflow as follows:

1. Design the database
2. Create a data dictionary (may be done at the same time as #1)
3. Wizard up an application (perhaps repeatedly)
4. Once you have a base app you're happy with, begin to change it:

In this article I'll start laying the groundwork for database design by discussing the various database options available to Clarion developers.

Flat file or relational

To some degree your design choices will be constrained by the kind of database you already have or choose to build. The most common choice is between flat file and relational (typically SQL) databases.

The term "flat file" covers everything from a simple, comma delimited text file to massive collections of indexed text and binary information such as TPS files. The key to "flat file" is the lack of any intelligence associated with the storage mechanism itself. A flat file database doesn't tell you what you can and can't do with the data; it's simply a repository.

Flat files

Clarion offers support for a variety of flat file formats:

- ASCII
- Basic
- Btrieve
- Clarion (DAT)
- Clipper
- dBase III
- dBase IV
- DOS
- FoxPro/FoxBase
- TopSpeed (TPS)

Many Clarion developers use TPS files for their applications' primary data storage, and a few still use the older Clarion DAT format. Both DAT and TPS files contain a moderate amount of information about the structure of the file itself; if your application's definition of the file layout differs from the physical file layout, your application will throw an error.

TPS files have a number of other advantages. They are relatively compact, they can be viewed and repaired using separate utility programs, they are sharable, they can be easily copied from one location to another, they can be encrypted, and multiple logical files can be contained in one physical file.

The primary disadvantage of TPS files is that they are not easily accessed outside of the Clarion world – you have to use SoftVelocity's ODBC driver.

In the early years TPS files had some reliability issues; these days TPS problems, if they occur, are usually network issues not TPS issues per se. Check out the TPS Troubleshooting FAQ for more.

You're on safe ground using TPS files for standalone applications and smaller networked databases (where the network itself is reliable). The maximum size of any one TPS file is two gigabytes.

Relational databases

A relational database system isn't just a file containing some data, it also actively manages access to that data. Most often relational databases are also SQL databases. SQL stands for Structured Query Language, which is an English-like syntax for selecting and updating data. Note that although most relational databases are SQL databases, SQL itself does not necessarily make a database relational. For instance, MySQL, particularly in earlier incarnations, is really a flat file SQL database – you can use SQL statements to work with the data, but some versions of MySQL don't define or enforce relationships between tables.

A true relational database is capable of enforcing constraints. If you have a database in which names can have multiple addresses, the database may prevent you from adding an address which is not associated with any name, or it may automatically delete all related addresses whenever you delete a name. A relational database can be made to enforce data rules so tightly that it's virtually impossible for any application to corrupt the data. On the other hand, flat file databases are relatively easy to mess up. And if you store your data in something as primitive as a comma delimited file, and any user can hack it with Notepad.

Clarion's support for relational databases includes:

- SQL/ODBC
- MSSQL
- SQLAnywhere
- Oracle

ODBC, like SQL, is a very old standard for database access. ODBC is a driver-based system. If your application can talk to ODBC, then it doesn't need to know how to talk to your back end database. All you need is an ODBC driver that can interpret your application's ODBC commands and take the appropriate action on the database.

Additional databases Clarion developers frequently access via ODBC include MySQL, PostgreSQL, and Firebird

The Clarion file driver layer

ODBC lets you deal with a variety of databases without changing your code, and that's a good thing. Similarly, Clarion's own file driver technology lets you work with ODBC, SQL, and flat file databases without changing your code. And if you're creating a new application using the ABC template chain (which is what you ought to be using for new

development) you'll also be using the ABC class library on top of those other layers.

Let's say you embed some code to retrieve a record from a MySQL database. You call some ABC methods, and some of that code uses Clarion's file access statements (SET, GET) which the Clarion driver translates into ODBC statements, which the MySQL ODBC driver translates into MySQL's SQL dialect. On the return trip the MySQL ODBC driver returns the data in an ODBC-compliant format, which the Clarion file driver layer translates into a Clarion data structure. Presto, there's your data!

That seems like a long and tortuous path just to get a bit of data, but in fact it all works quite quickly and smoothly.

ADO and ADO.NET

ADO stands for ActiveX Database Objects, which is Microsoft's successor to ODBC. Although Clarion does ship with a bunch of ADO classes, they're not interchangeable with the standard Clarion way of handling databases, and so haven't caught on. But they can be very handy for dealing directly with record sets. See Tom Ruby's article on the cCwADO class for an example.

ADO.NET is, as you'd expect, ADO for the .NET platform (although in many ways it's quite different from ADO). In Clarion.NET you'll have the option of using ADO.NET indirectly, via Clarion language statements, or by calling the ADO.NET classes directly. Bob Zaunere commented on this last year on the SV web forum:

> Clarion.Net provides the flexibility to choose from three data access methodologies; standard Clarion syntax, our DAL, or plain ADO.Net. The standard Clarion data access methods (NEXT, PREVIOUS, etc) use interop built-in to the RTL to call the Clarion win32 drivers, this approach is required for access to the ISAM files like TopSpeed and Clarion formats. Using this approach you could for example fill a Queue using standard Clarion access methods, and use it as the data source for a datagrid. In the future we might decide to port some of the win32 drivers to managed code, or to create a generic ADO.Net provider for ISAM files, or to write an ADO Provider that utilizes the IP driver/server, i.e.. the ADO commands are sent from the IP driver (ported to .Net) to the IP data server. The second option uses our Data Access Layer classes (DAL), which wraps ADO.Net and emulates the standard Clarion language syntax. This approach works for all

backends that have an ADO Provider. The third approach available is to use straight ADO.Net, and then the code looks just like C# using Clarion syntax, i.e no curly braces etc. And of course these options are not mutually exclusive.

Future directions

Bob Zaunere has posted that a future version of Clarion.NET will support Microsoft's LINQ technology. I've played with LINQ and it's pretty cool stuff. LINQ extends the concept of a common query syntax beyond databases to XML files and even to the code that makes up your application.

Back ends, client-server systems and relational flat file databases

Clarion muddies the water somewhat in that it has for many years made it easy for developers to create what are effectively relational database systems using flat files. That is, Clarion is fully capable of supplying the necessary code to manage database relationships, in the event that the back end database doesn't have this capability.

Lest this become too confusing, let's review a few terms as they apply to Clarion development:

- Client – an application that uses a shared, server-based database
- Server – an application that manages access to a database on behalf of a client
- Relational database – a way of storing data and enforcing rules and data integrity. Relational databases are a combination of data and a data server.
- Flat file – a simple way of storing data, no server required

Here are a few ways you can use Clarion to cover the spectrum of databases from flat file to relational database. Your application may:

- Read/write simple text files
- Read/write single user, more efficient binary files (such as DAT or TPS files) on your local computer
- Read/write shared, more efficient binary files (such as DAT or TPS files) on a server
- Access a single user or shared relational database which gives simple access

to data without enforcing relationships or data integrity
- Access a single user or shared relational database which tightly controls data relationships and integrity

And as I said earlier, Clarion gives you the option of deciding whether to enforce data integrity on the server (when the server is capable of it), or on the client, or in some combination.

And if all that isn't enough, you can mix as many of these approaches and databases as you like within a single application!

Making sense out of databases

If you don't have a database in mind, what should you choose? I think for most users there are only two real choices, despite the many possibilities.

If you're talking about a multi-user application, you should think seriously about SQL. Whether you use a native driver or an ODBC driver isn't usually a big issue. When using SQL you should also enforce relational integrity on the server side as much as possible. Yes, Clarion can generate code to do that for you on the client, but there are two advantages to keeping this code on the server:

1. Because the relational integrity code runs on the server, less data has to travel over the network, and your client app should run faster.
2. Other applications accessing the data will be subject to the same data constraints. If only your Clarion application knows and obeys the rules, somebody else's application can still trash the data.

SQL also offers some tremendous opportunities for batch processing. You can use a single SQL statement to update massive amounts of data on the server in a very short period of time; moving all that data across the wire will take much longer and suffer greater exposure to any network problems. You can easily execute SQL statements directly in Clarion using PROP:SQL. And SQL servers typically scale well; you can build much bigger databases, handling more users, than you can with flat files.

The downside to SQL is the extra administrative work. You need to install a server somewhere on the customer's system, create the database, and set access permissions. Most likely you'll want to adjust the server's configuration for maximum performance (a process that can be more art than science), and you may need to use a server-specific

utility for backups.

If you have no foreseeable need for SQL, high levels of transaction processing, or many users on a network, or if you want to ship self-contained applications that have no need for a separate SQL server, then TPS files are probably your best bet. (In theory there are embeddable SQL engines you could use with Clarion, but I don't know of anyone who's done this - if you've done it or know someone who has, let me know.)

TPS files are also a lot more convenient for shipping demo software, or for any situation where database server administration skills are in short supply.

You can always start with TPS and migrate to SQL as needed. Just keep in mind some of the conversion issues and in particular stick with SQL-ready data types.

Beginning the design

Once you've settled on your database technology, you can start tackling the design. Or can you? I've alluded to some of the differences between SQL and flat file systems, and there are other differences between the various SQL platforms. It may be that as you design your database you'll come across specific requirements that can only be met by a certain databases. If you have unusual requirements, you may need to do a certain amount of database design before you can make a final decision.

In any case, I suggest you read the next installment in this series (available soon) before you make that decision.

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

Replicating IDLE: Throwing Users Out

by Steven Parker

Published 2006-10-12

"When the user hasn't done anything for ***x*** minutes (or seconds), I want to close the window. How can I do this?" This question comes up fairly frequently. As is so often the case using Clarion, there are a number of ways to accomplish the goal of throwing the user out (there are several non-Clarion ways to accomplish this but they are not germane here).

The Clarion language provides one obvious solution, the `IDLE` statement. As the Language Reference states:

> `IDLE` Arms a procedure that periodically executes….
>
> An `IDLE` procedure is active while `ASK` or `ACCEPT` are waiting for user input. Only one IDLE procedure may be active at a time. Naming a new `IDLE` procedure overrides the previous one. An `IDLE` statement with no parameters disarms the `IDLE` process.

Unfortunately, while `IDLE` looks tailor made to the task, as I observed in Replicating IDLE: All Quiet on the Keyboard?, it isn't a usable solution:

> `IDLE` has been broken since (at least) Clarion for Windows 1.5. And, it is broken badly. For example, the documentation states "Only one `IDLE` procedure may be active at a time. Naming a new `IDLE` procedure overrides the previous one." This is true in 16 bit; false in 32 bit (new `IDLE` procedures in 32 bit are unlikely to execute at all) .

In 32 bit programs, an `IDLE` procedure on the app frame works (i.e., an `IDLE` procedure called from the frame). But no other `IDLE` works; an `IDLE` procedure called further down the calling chain not only doesn't override any `IDLE` procedure that may have been called in the frame, it simply doesn't work. I wrote "Replicating IDLE" in 2001; at that time, not only did the `IDLE` procedure respond only to keystrokes in the frame procedure, the called procedure needed to be in the same module as the frame.

Perhaps it would be less unfair to SoftVelocity to say that there is a major documentation bug. In fact, I think that most of us who have been around for a while have just stopped using `IDLE` and consider it a 16 bit relic.

As I said, Clarion typically provides multiple paths to application nirvana and in this case, another path is most definitely required.

But Wait!

"But," you may be thinking (I certainly did) "if I can start an `IDLE` procedure on the frame, I *could* post events to the frame and reset the frame's `IDLE` procedure." Also, SoftVelocity introduced the `NOTIFY` statement (at least for 6.x users) since "Replicating IDLE" appeared. Not only has `NOTIFY` been well received within the community (i.e., it works, works well and works easily), it too would appear appropriate to the job.

The reason I rejected the `IDLE` approach in 2001 and the reason I continue to reject it is that it is both too complicated and too insensitive.

First, "too complicated." Yes, posting events to the frame is easy (see, for example, Sidebar Menus). But, in this case, should the inactivity period be up, an event has to be posted back to the desired procedure. So, the frame needs to know the correct procedure to be affected. If that procedure is not the first procedure on the thread, it must be on top. If it is on a different thread, `THREAD()` must be known. Frankly, "complicated" seems a bit of an understatement when trying to post an event back to the correct procedure.

Second, "Insensitive." I am thinking that posting a user defined event back to a browse (that user defined event, in turn, calling `Post(Event:CloseWindow)`) is not especially troublesome. But what if the procedure to be closed is a data entry form? I have seen cases where partially completed forms have been saved by `Post(Event:CloseWindow)` and required fields have been left empty. And what about "Action on aborted Add/Change?"

No, posting events to the frame, having the frame's `IDLE` procedure check whether action is required and, if it is, posting an event back to the "caller" is just too scary. Of course, an additional issue is that every procedure in an application would have to communicate with the frame to prevent the `IDLE` procedure call or every procedure would start receiving spurious events. If a procedure did not post events to the frame, the frame's `IDLE` would hit its timeout and … what would it do? "Event bloat," not elegant at all.

TakeEvent

I feel, based on what I learned researching "Replicating IDLE," that the embed points for the `TakeEvent` and `Event:Timer` methods are what I need. They have the virtue of not limiting me to only one `IDLE` at a time (I can use these embeds in any or all open windows). Neither do these embeds limit me to calling only procedures (I can write in-line code or call a local method or routine) or only procedures without parameters. All of these are restrictions of the `IDLE` statement.

This flexibility is especially useful in throwing users out because a simple

```
Post(Event:CloseWindow)
```

or

```
Post(Event:Accepted,?Cancel)
```

(in the case of forms) is all I really need to do. I don't need to call a procedure at all to throw a user out.

The solution I came to five years ago was to use `TakeEvent` to see whether the user had been active or not. If the user had been active, I reset my timeout. For example:

```
Timeout = Clock() + (NumberOfMinutes * 6000)
```

In this code I compute the actual time at which to throw the user out (see `UpdateCustomers2` for an example of setting the time of day at which to kick the user out – note that I convert the number of minutes before time out to hundredths of a second, the same units that `Clock()` uses). Or I use a countdown timer. This code:

```
Ticks = DefaultNumberOfTicks
```

resets the `Ticks` value (assuming `DefaultNumberOfTicks` is already in Clarion Standard time – see Marking Time, Part 1; `UpdateCustomers1` is an example of a countdown timer).

Because any event, any mouse click, keystroke, anything calls `TakeEvent`, the `TakeEvent` embed really is the ideal place to check whether an event has occurred and what event it is. If a user event occurs, like a keystroke or mouse click, I want to restart the time counter.

However, because timer events also call `TakeEvent`, `Event:Timer` will too. I don't want to reset my counter(s) just because the timer fired. So, my `TakeEvent` code is:

```
If Event() <> Event:Timer
  ! reset counter
End
```

Doing the dirty

I use the window's `Timer` method to see if the inactivity period has expired and then I act as needed. This part is easy.

In a browse (in the sample app, see `BrowseCustomers`):

```
If Event() = Event:Timer
  Timeout -= 1
  If Timeout = 0              ! countdown
    0{Prop:Timer} = 0   ! to zero
    Message('Timed out. Substitute IDLE is ' & |
            'working.','IDLE()',ICON:Exclamation)
    0{Prop:Timer} = 100
    Post(Event:CloseWindow)
  End
End
```

kicks the user out.

In a form (see `UpdateCustomers1` or 2), I prefer to mimic the user's clicking the Cancel button:

```
If Event() = Event:Timer
  If Clock() > Timeout   ! clock .GT. computed time
    SELF.CancelAction = Cancel:Cancel
    Post(Event:Accepted,?Cancel)
  End
End
```

Note that I also explicitly set `SELF.CancelAction = Cancel:Cancel`. This ensures that my app doesn't hang with one of the "Are you sure?" messages.

Who Did What?

If an event is posted and it is not a user event, i.e. the timer has ticked over, I can check whether the inactivity period has or has not expired in `Event:Timer` (obviously, the window will need a timer). If it has expired, I can act.

Here is where it gets harder. Much harder.

In a browse's `TakeEvent` method:

```
If Event() <> Event:Timer
  ! compute or reset
End
```

is sufficient. `BrowseCustomers` shows this technique.

However, in a form, `TakeEvent` receives a number of events, many of which are neither timer nor user events. These include `OpenWindow`, `Resume`, `Sized`, `Selected`, `Suspend` and `GainFocus`. In fact, in 5.5 (I haven't tried this .APP in 6.x) several of these events fire constantly. As a result, regardless of the method used to determine inactivity, the inactivity computation never trips. (`GainFocus`, particularly, fires a lot – a *lot* – given that this form is called by a `START`ed procedure, I don't suppose this is really surprising; but it is a royal PITA to have my timer reset every time my CPU time slices back to my form).

Checking `KeyBoard()` doesn't work. After the first keystroke, `KeyBoard()` always returns `True`. Again, the condition is never satisfied. Alerting keys and checking those, similarly, gives no joy. Take my word in this. Please.

To make a long story short:

```
Case KeyCode()
Of 33 to 128 orof MouseLeft orof TabKey
  ! compute or reset
  SetKeyCode(0)
End
```

seems to do just what I want. The trick is in `SetKeyCode(0)`. This ensures that the next cycle of the `Accept` loop doesn't trigger `Case KeyCode()`. As they say, "salt to taste," you may want to check different key codes (I just grabbed the basic alphameric keys for this example).

Try either `ThrowOut1` or `2`. Tab around. Click on each field in turn. It will not time out. Enter something in each field but do not press a button. Fold your hands in your lap. A few seconds later (the default in the sample app is five seconds), the form disappears! And *that* is just what I want.

NOTE: For purposes of this demonstration, only run one browse at a time.

Summary

Kicking a user out ought to be easy. It is, if you are kicking her/him out of a browse or, perhaps, a "simple" window. It gets more … challenging when data entry forms are the target of the exercise.

The essentials are easy. I know I have to know:

- How long the user is allowed to be inactive
- What action to take (different actions for different template procedures)
- When to reset my counter

Resetting the counter, however, isn't so easy as I thought.

```
If Event() <> Event:Timer
  ! reset counter
End
```

is counterproductive in a form. And that was my big lesson from this: there are more events in forms than are dreamt of in my philosophies.

Read part two.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Clarion 101: Understanding the IDE, Part 2

by Dave Harms

Published 2006-10-05

In Part 1 of this new series I provided an overview of several key components of the Clarion application generation system, including the integrated development environment (IDE), the Dictionary Editor, the Application Generator, the template language and the code embed system.

Now, as important as those core components are to the Clarion system, they're far from everything that defines Clarion. In this article I'll talk about some of the other vital aspects of Clarion, including:

- The Clarion language, procedural and object-oriented
- The ABC class library
- The database grammar
- The database drivers
- The browse/form concept

The compiler

Before I get into the language, a word about the compiler. In the early DOS days, Clarion programs were pseudocode. That meant you compiled your Clarion source to an intermediate format, which had to be run through an interpreter. A later DOS version (the infamous Clarion Database Developer, which nearly sank the good ship Clarion) produced actual DOS executables (but not much else of worth, at least up until release 3.009), and of course the Windows version of Clarion has always created true Windows EXEs and DLLs

(and LIBs).

Clarion's Win32 compiler technology is an evolution of the JPI compiler technology. JPI was started by Nils Jensen, one of the founders of Borland, using technology Philippe Kahn decided he really didn't want. JPI and Clarion Software merged in 1992 to form TopSpeed Corporation. You can read more Clarion history at Wikipedia.

The Clarion language

The guy you want to blame for the Clarion language is Bruce Barrington. Okay, blame is a strong word. Maybe "thank" would be better. I only blame him for the daft notion that `ALL CLARION KEYWORDS SHOULD BE UPPER CASE`.

Clarion is an eloquent business application language, although in recent years it has strayed in small ways from Barrington's dream of a language that is "powerful and expressive, yet easy to read and maintain." Clarion is still powerful and expressive, far more so than in the early 1980s, and with that power comes additional complexity. The addition of object-orientation to the language has arguably decreased readability.

Of course, ol' Bruce thought all caps was readable, so maybe not much has changed after all.

Here's a Clarion version of the ubiquitous "Hello World" program. (Incidentally, the first ever "Hello World" program was written by Brian Kernigan in the B language, and the B language of course gave birth to the C language. We now return you to your regularly scheduled programming language.)

```
program

    map
    end

    code
    message('Hello World')
```

All Clarion programs begin with a `PROGRAM` statement and are case insensitive, which is why I can write `PROGRAM` or `program` or `proGRam` and my sloppy little app will still compile. I tend to write lowercase for speed, or mixed case (or camel case) for readability.

A hybrid language

For many years Clarion was strictly a procedural language, and you can still use it for just procedural code. Here's another tiny source listing that illustrates the use of procedures:

```
program

    map
      DoSomething()
      DoSomethingElse()
    end

    code
    DoSomething()


DoSomething     procedure
    code
    message('Something')
    DoSomethingElse()

DoSomethingElse procedure
    code
    message('Something else')
```

This, in a microcosm, is how a procedural application is set up. The `MAP` statement defines the procedure prototypes, which are declared in the source listing (there are ways of splitting applications up into multiple source files, but that's beyond the scope of this article). The first `CODE` statement is the application's entry point, and in this case it calls the DoSomething procedure, which in turn calls the `DoSomethingElse` procedure. Procedures can contain their own data, and can receive and return parameters. Procedures can also contain their own local procedures, called routines, which are available only to that procedure.

In general, you don't see a whole lot of code being called at the application entry point (that first `CODE` statement). Usually there's nothing more than a bit of setup code and then a call to a procedure that is the application's main menu.

Any data declared before the first `CODE` statement is *global* data. Global data is visible throughout the program and is basically evil. Sometimes it's a necessary evil, and sometimes it's an unnecessary evil. The idea is that you want your data shared as little as

possible among procedures, because the more code that can alter your data, the more likely it is to be altered in unpleasant ways.

There is no global data shown in the above example because it is, as I said, basically evil, and that is a good little program. In a real-world Clarion application you'll find files/tables and a few important variables declared globally.

For more information on writing Clarion code, see the Getting Started – Clarion Code topic.

As I said earlier, Clarion is a hybrid language. With Clarion (for Windows) version 4 the language grew some object-oriented extensions. Clarion is not, however, a pure object-oriented language. In pure OOP, everything, including the application itself, is a class, and that class will typically have a *method* (the OOP name for *procedure*) with a magic name like main. In Clarion the application is an application, not a class, but you can create as many classes as you want and use them any which way you like. Here's an extremely simple version of the above program, rewritten to use a class:

```
program

    map
    end

SomeClass       class
DoSomething       procedure()
DoSomethingElse   procedure()
                end

    code
    SomeClass.DoSomething()


SomeClass.DoSomething     procedure()
    code
    message('Something')
    self.DoSomethingElse()

SomeClass.DoSomethingElse procedure()
    code
    message('Something else')
```

You'll notice that `SomeClass` is declared globally, and you already know that global

data is evil. Global classes are not as worrisome as simple data, however, because you can if you wish make class data private and exercise control over how the data values can be altered.

For more information on Clarion's object oriented capabilities, see The ABCs of OOP.

The ABC library

Clarion would be in a world of hurt right now if not for object orientation, and not just because every other language on the planet can do objects.

In Part 1 I explained how Clarion uses templates to generate application code. For procedural applications, that means that the templates generate *all* the code. Every loop getting records for a browse, every line of record update log, all the menu handing code etc. A *lot* of this code is exactly the same each time, except for minor changes to data.

Other than all that code duplication there's nothing really wrong with generating all the code, since the templates handle the grunt work. Except for this one thing:

Templates can be a bear to write.

Templates are a particular problem because they're a mix of template language code and Clarion language code. Among other things, that means you can't test templates; you have to use them to generate some code, and then you can test the code. And because templates not only mix template language and Clarion language statements, but use template procedures (#GROUP statements) to avoid code duplication, it's terrifically hard sometimes (if not most times) to just read the Clarion code in a template. Here's some template code for handling backspaces on a browse locator, in procedural Clarion:

```
IF KEYCODE() = BSKey
  IF %BrowsePrefix:LocatorLength
    %BrowsePrefix:LocatorLength -= 1
    %BrowsePrefix:LocatorValue = |
       SUB(%BrowsePrefix:LocatorValue,1,|
       %BrowsePrefix:LocatorLength)
    %BrowseLocatorName = %BrowsePrefix:LocatorValue
    #SET(%ValueConstruct,%False)
    #FOR(%KeyField)
      #IF(%ValueConstruct)
```

```
        #IF(%KeyFieldSequence = 'ASCENDING')
    CLEAR(%KeyField)
        #ELSE
    CLEAR(%KeyField,1)
        #ENDIF
      #ELSE
        #IF(%KeyField = %BrowseLocatorName)
          #SET(%ValueConstruct,%True)
        #ENDIF
      #ENDIF
    #ENDFOR
    %InstancePrefix:LocateMode = LocateOnValue
    DO %InstancePrefix:LocateRecord
  END
```

Some of the Clarion code is dependent on the contents of template symbols like `%KeyField` and `%BrowseLocatorName`; some code will be generated or not generated based on the value of other symbols (such as `%ValueConstruct`). And this is one of the more readable template fragments.

While templates greatly improve the end user's (that is, the developer's) productivity, they can easily become a nightmare for the template author. And that's just what happened to Clarion for Windows. By the time Clarion for Windows 2.0 was released, the original template set, called the Clarion template chain (often referred to now as the Legacy templates) was showing some serious strain under the increasing feature set expected by Clarion developers.

The solution to this problem was to split the Clarion code (or as much of it as possible) out of the templates, and into an object oriented source code library. This library is called ABC, for Anything But Clarion [templates]. No, actually ABC stands for Application Builder Class library.

The role of the templates then changed from controlling the generation of all of the code to generating code that made calls, as needed, into the ABC library. In essence the templates became a programmer's front end for ABC.

So why couldn't this have been done with the Clarion templates as well, using procedural code? I think this is best explained by looking at the browse procedure. Browses, in Clarion at least, are inordinately complex creatures. There have been a few efforts over the years at creating generic procedural browses in Clarion (most notably by Todd Seidel) but ultimately the problem is that you don't have enough control over what's going on inside

that generic code. You always need to get inside and muck about with some part of the plumbing, and procedural code doesn't have a good mechanism to allow this.

Object oriented code, however, does have such a mechanism, and it's called the virtual method. Virtual methods are a little like callback procedures, only much more powerful because they live in an object-oriented environment. And as you may have guessed by now, they are the enabling mechanism for embeds in ABC applications.

ABC applications are almost entirely OO applications, and just about everything you do happens inside some class's method, whether you realize it or not. When you embed some code, what typically happens is your code gets generated into a virtual method, which is like a copy of an existing method in the class in question. But instead of the original method being called, your method is called instead (and frequently the template-generated code that wraps up your embed will also call the original method – this is what the Before parent call and After parent call embed points signify).

For a whole lot more on templates and embeds see the Embeds, using topic.

The ABC library

I've already mentioned the ABC library, introduced with Clarion 4. About 75 classes are documented in the Clarion help, but the total number of specifically ABC classes is more than double that, and if you go searching in your libsrc directory you'll actually find around 400 different classes of all kinds.

There are a handful of classes you'll find it just about impossible to avoid. These include:

- `BrowseClass` – manages browses, in conjunction with a window definition
- `ViewManager` – parent of BrowseClass, manages file/view operations including sort orders and filters
- `FileManager` – handles retrieving and adding/updating data for a single file, one instance per file
- `RelationManager` – manages relationships between files, one instance for any file with relationships
- `WindowManager` – controls behavior for any standard window (menu, browse, form) – one instance per window.
- `ReportManager` – manages a report (one instance per report), derived from WindowManager.

There are a bunch of other important classes, but these are the ones you'll encounter most often.

There are two print books which look in some detail at ABC. Bruce Johnson's Programming in Clarion's ABC provides a lot of detail on the inner workings of ABC; Russ Eggen's Programming Objects in Clarion deals more with the theory of OOP, and also provides details on writing ABC template wrappers.

The database grammar

One of the hallmarks of Clarion, from its first DOS version, is its database grammar. Clarion uses a straightforward syntax to sort, retrieve, and update data: verbs include `SET`, `GET`, `NEXT`, `PREVIOUS`, `ADD`, `PUT`, and `DELETE`. This has become a bit muddled with the advent of ABC and its wrappers for the database statements, and you may find yourself mixing and matching. The nice thing about ABC is calling, say, the `FileManager`'s `Fetch` method and not having to write any error handling code because it's already in the `FileManager` class.

The database drivers

Clarion allows you to mix and match data sources in a single application by virtue of its system of database drivers. The idea is you can use a single syntax to access any of a number of flat file and SQL databases without having to modify your program code. This is generally true, but because different back ends (particularly SQL databases) offer unique features, you may find that you end up targeting one particular database anyway.

The good news is that if you are migrating from a flat file database (DAT or TPS files, for instance) to a SQL database, you may be able to do so relatively painlessly.

The page-loaded browse/form concept

The page-loaded browse/form concept is something all experienced Clarion developers come to think of as normal, but it's far from universal in business application development.

Browse/form says I'll show you a list of records, and you'll use a button or other control to insert, change or delete records, most likely using another window (the form).

And not only will the browse show you your records, it will do so only a page at a time. That means that if you have a bazillion records in your table, only one page worth will be retrieved at a time. That saves both time and resources.

Closely allied with the browse/form concept is the idea of optimistic concurrency. Clarion has been a multi-user client/server environment from the beginning, which means that more than one user could be trying to update a given record. Pessimistic concurrency says someone else is likely to want to update the record I update, so I'll lock it until I'm done with it. Optimistic concurrency says conflicts are unlikely, and this is Clarion's approach. Clarion will retrieve the record, leave it unlocked, and allow you to make your changes. Before the record is written back, some of that magic code Clarion writes for you checks the existing record to make sure no one else has changed it. If someone has changed it, you're shown the changes and given an opportunity to make your own.

There are many alternatives to the standard Clarion approach of page-loaded browses and forms with optimistic concurrency. Some, like loading all records into memory, are template options. There's support for edit-in-place as well.

Putting it all together

Here then are the key Clarion features I've discussed in Parts 1 and 2:

- The Application Generator
- The Dictionary Editor
- The template language
- The embed system
- The Clarion language
- The ABC class library
- Database grammar and database drivers
- The browse/form concept

All of these exist in the context of the IDE itself. How then do you put all of these pieces together to create an application?

Here's a typical workflow:

1. Design the database
2. Create a data dictionary (may be done at the same time as #1)
3. Wizard up an application (perhaps repeatedly)
4. Once you have a base app you're happy with, begin to change it:
    - Modify the appearance of menus, browses and forms via window and report designers
    - Modify behavior by setting template prompts.
    - Embed code to add new functionality
    - For extra productivity, create templates to automate addition of new features/behaviors

You may also want to use third party products at just about any stage of development, but that's beyond the scope of this article.

In the next article in this series I'll begin describing this workflow, starting with database design and the data dictionary.

Suggested reading

- Getting Started – Clarion Code (topic)
- Embeds, using (topic)
- ABCS of OOP series
- Tom Giles' article 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

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

Search database rebuilding now

Direct link

Posted Friday, October 27, 2006 by Dave Harms

Search problem solved - I'd forgotten to create a couple of directories. The search database is being rebuilt and should be up to date in a few minutes.

Migration glitch #1 - search feature

Direct link

Posted Friday, October 27, 2006 by Dave Harms

Okay, so this isn't exactly the first migration glitch. But the search database doesn't seem to be rebuilding properly. Stand by...

Google code search exposes passwords

Direct link

Posted Wednesday, October 11, 2006 by Dave Harms

On Monday I blogged about code search sites. Today Mark Riffey pointed out this blog entry from the Joomla folks about Google Code Search exposing passwords and other private information from PHP files:

> It has come to our attention that Google has released a new product, Google Code Search, that is capable of indexing and crawling through archive files stored in the public directories of web servers. We are reporting this as a security advisory because we have discovered that some site administrators are storing archives / backups of their website in the web root. Because of this, Google Code Search is able to crawl the archives and read unparsed PHP files as if they were plain text. This has resulted in the disclosure of some sensitive information including MySQL passwords and SMTP credentials.

As Mark points out, whether or not you're using Joomla it's important to verify that you're not accidentally exposing any private information on a web server.

Code search sites

Direct link

Posted Monday, October 09, 2006 by Dave Harms

Last week Ken Bame pointed out Google's new code search page. Unfortunately Clarion is not one of the languages listed, not surprising given its low market penetration. On the surface, Google's effort looks a bit less sophisticated than Krugle. But Krugle doesn't list Clarion either (I requested it be added, but never heard back from the developers).

You can try searching Google's site for .CLW files but that won't get you far because in Visual C++ that extension indicates a Class Wizard File.

Oh well.

CapeSoft + single board computer = CapeFox

Direct link

Posted Wednesday, October 04, 2006 by Dave Harms

Several times recently Bruce Johnson has mentioned CapeSoft's work with single board computers/embedded systems. Bruce likes the Fox board, a complete Linux system on a single 2.6 x 2.8 inch board. For more on CapeSoft's use of this board including some HowTos see the CapeFox site.

GenWise and dual core speedups

Direct link

Posted Tuesday, October 03, 2006 by Dave Harms

Peter Rakke at RADVenture reports on how AMD's utilities can improve performance for some applications on dual core systems. Peter had a GenWise project that took five

minutes to create; after running the ADD Dual Core Optimizer the same task took only 18 seconds. Clarion compiles were also signficantly improved.

Peter has been showing me some of GenWise's impressive capabilities, and I expect to have a review in the mag in the near future. Unlike RADVenture's Fenix, which is a set of Clarion templates to generate ASP.NET applications, GenWise is a standalone development environment which uses in-house template technology, NHibernate ASP.NET 2.0 WebForms.

A hand coder's Clarion.NET?

Direct link

Posted Monday, October 02, 2006 by Dave Harms

I just responded to a comp.lang.clarion thread about text relevancy searching in which I mentioned the Lucene search engine (an older version of which powers ClarionMag's search feature). Mark Riffey was the one who put me in mind of Lucene, as he'd just pointed me to CLucene, a C++ version of this very popular search engine library. Interfacing Clarion and C++ isn't easy, but it can be done. Maybe some enterprising third party developer will make up a front end for Clarion developers.

And then you have Lucene.NET. Okay, interfacing .NET and Clarion isn't appreciably easier than calling C++ code from Clarion. But it too can be done.

But using Lucene.NET with Clarion.NET? Same as using it with any other .NET language. Easy peasy.

I expect the full version of Clarion.NET with templates is still some ways off, since no one I know has even seen a demo. But a hand coder's edition would already be useful. Consider Lucene.NET. You could easily enough do a small hand coded Clarion.NET assembly to create a Lucene index of some data, and use the simplest of interop calls (following Wade Hatler's example) to have another bit of Clarion.NET code load up a table with query results.

Apparently SV doesn't want to release a hand coder's edition first for fear of confusing users who are expecting the full AppGen edition. Okay, that's a risk. In my opinion it's one

worth taking, and it would also help flush out a whole bunch of bugs in advance of the AppGen edition.

If you can use a hand coder's edition of Clarion.NET, let SoftVelocity know!

C6 and Vista

Direct link

Posted Monday, October 02, 2006 by Dave Harms

There was a short discussion in comp.lang.clarion late last week about whether Clarion's antiquated 16 bit IDE will run on Windows Vista (formerly known as Longhorn), which is scheduled for release next month to business customers. It seems pretty clear you can't run the IDE on 64 bit Vista (and I assume you can't on 64 bit XP either since that OS lacks NTVDM, the subsystem that runs 16 bit apps). But if you don't have a 64 bit system you'll be running 32 bit Vista, and several Clarion developers have reported running the Clarion IDE on that version of Microsoft's new OS.

And in any case, Clarion 6 *applications* will run fine on Vista, since they're all 32 bit. Clarion 5.5 was the last release to support 16 bit applications. So at least no one can say that the current release of Clarion creates apps that won't run on Vista.

Will the release of Vista spur SV to get C7 out the door? Hard to say. But if Clarion developers are content with 32 bit Vista, it's not a requirement to keep the troops productive. Note also that the Clarion roadmap is suitably vague about any actual delivery date; C7 is "*currently scheduled* to be completed and released this year (2006)" (emphasis added).

Update: I'm told that VMWare is another way to run 16 bit apps on 64 bit Windows.