Clarion Magazine

Clarion News

- » Fomin Report Builder 3.07
- » xFText v2.8
- » J-Skype Price Increase This Week
- » xReportPreview v3.5
- » DriveInfo Updated
- » MapPoint Templates
- » CoolFrames Beta 1.0
- » 1stIconDesign Sale
- » Gitano Christmas Special
- » J-Fax, J-Skype, and J-Spell Bundle
- » One Week Left In Capesoft Amnesty
- » Class Anatomy 1.3.0
- » DriveInfoClass Bug Fix
- » J-Spell Adds Portuguese
- » Clarion Desktop 2.05
- » GTL 6.26 Released
- » Clarion Desktop v1 Expires
- » Huenuleufu Moves To .COM Domain
- » .MOBI Domains At Oak Park Solutions
- » Clarion Desktop 2.00 (Gold)
- » EasyOpenOffice 1.03
- » NeatMessage 2.00 Beta
- » Dr. Watson Analyzer
- » SimSoft Clarion Desktop Compatibility
- » Firebird 2.0 Released

- » AppScanner Lite Debugger
- » Clarion Desktop 1.35 Adds Downloads, Video
- » IAlchemy Office Schedule
- » ProImage Demo and Sale Price Extension
- » J-Fax 1.32
- » J-Skype 1.10
- » SysInternals Now At Microsoft
- » SQLShell Released
- » Oak Park Wins Hosting Award
- » Icetips Previewer 2.4
- » Data Interoperability Community of Interest Handbook Released
- » ProDomus Adds Clarion Desktop Support
- » ProDomus PD Version Notes and CLE
- » SV Driver Sale Reminder
- » Huenuleufu RSS Feed
- » PostgreSQL Support in FM3
- » CapeSoft License Amnesty in November
- » Clarion ProImage Released

[More news]

Clarion 101

- » Clarion 101: Understanding Keys and Indexes

[More Clarion 101]

Latest Free Content

[More free articles]

Clarion Sites

Clarion Blogs

Latest Subscriber Content

Printing A Tree From A Page Loaded Browse

Previously, David Podger and Deon Canyon have showed how to implement a page loaded tree browse. Now it's time to print that tree.

Posted Thursday, November 30, 2006

Controlling Controls

Steve Parker has a customer browse which must indicate whether a customer has an email address, and if the email address is present, allow the user to send an email. But as usually seems to happen, the obvious solutions don't apply.

Posted Thursday, November 30, 2006

How To Use The Debugger, Part 2 (video)

Part 2 of Richard Rose's Camtasia presentation on how to use the debugger is now available to ClarionMag subscribers by special arrangement with the UK Clarion User's Group. This is a 10 minute, 25 meg AVI file. If you have trouble downloading please delete any partial downloads and do a Save As.

Posted Tuesday, November 28, 2006

Solving Problems With Finite State Machines

Finite State Machines are a useful technique for many programming tasks, including text searching. If you've ever done any complex text parsing you may even have used one of these without knowing it. John Christ explains.

Posted Friday, November 24, 2006

Adding Arrays To Generic Queues With HOWMANY

Alan Telford uses Excel to view queue data, as explained in an earlier article. That version of his generic-queue-to-CSV exporting procedure couldn't handle arrays in queues, but Clarion 6's new HOWMANY function makes arrays in generic queues usable.

Posted Friday, November 17, 2006

Clarion 101: Understanding Keys and Indexes

Keys and indexes are vital to getting good performance out of your database. But what is a key, and what is an index? And how do you use them? Bruce Johnson explains.
Posted Thursday, November 16, 2006

How To Use The Debugger (video)

Richard Rose's Camtasia presentation on how to use the debugger is now available to ClarionMag subscribers by special arrangement with the UK Clarion User's Group. This is a 26 minute, 33 meg AVI file. If you have trouble downloading please delete any partial downloads and do a Save As.
Posted Tuesday, November 14, 2006

October PDF updated

A nagging character set issue with the new server, which resulted in some odd-looking punctuation characters, appears to be resolved. I've updated the October PDF with the affected articles.
Posted Monday, November 13, 2006

Threads: When START Starts

You may think that when you START a procedure, it immediately starts. This is not, however, true. Steve Parker shows how to exercise control over when a STARTed procedure actually begins to run on its own thread.
Posted Friday, November 10, 2006

Signing Your Applications, Part 2

Increasingly Windows security makes it advisable to be able to sign not just your installer but your actual EXEs and DLLs. Jane Fleming concludes her two-part series on signing applications with two approaches to signing: wizard based and batch file based.
Posted Sunday, November 05, 2006

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

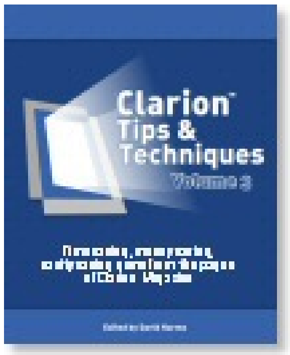
Printed Books & E-Books

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 3 - ISBN: 0-9689553-9-8
- 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

Fomin Report Builder 3.07

Fomin Report Builder 3.07 is now available. Changes include:ClarionDesktop compatibility; FM3 compatibility (there was a problem in "Local" application mode); Fixes to Firebird ODBC problems; FRB::Version STRING('3.07') external variable added; FRB classes ABC compliant (thanks to Russ Eggen) - no need to use FRBCheckExportList template in a Root DLL any more; "Fixed at the top of page" checkbox disabled for Page Header and enabled for Report Header band; CPCS compatibility module updated; Other tweaks and fixes up to date.
Posted Tuesday, November 28, 2006

xFText v2.8

xFText is a class and extenstion template that lets you add text and images anywhere in your application frame. Features include: Set global text margins; Set font attributes; 3D text imitation; Position offsets; Runtime text changes; Include bitmaps as resources. Updated demo and install kit.
Posted Tuesday, November 28, 2006

J-Skype Price Increase This Week

The first gold version of J-Skype will be released Thursday night this week, at which time the price will be increasing from $49 to $89. Anyone who has purchased the beta release (for just $49) will receive a free.
Posted Monday, November 27, 2006

xReportPreview v3.5

New xReportPreview v3.5: Two new methods. BeforeFlushPreview and AfterFlushPreview, called just before and after pulling queue to printer. You can find it in GlobalEmbeds; New properties. Original queue SELF.OrigPrnQueue, before printing includes names of original wmf-files. (without headlines); Demo application now include examples of using exporting to PDF, HTML, TXT. Limitation! Export without headlines; Updated demonstration program; New installation kit created by SetupBuilder 5.3; Bugfix: Using of template Pause Button; New example with source application; Updated Demonstration programs and Installation Kit for Clarion 6.x and Clarion 5.5 are available. xReportPreview is $59.
Posted Monday, November 27, 2006

DriveInfo Updated

The latest release of DriveInfo adds a template for ease of use. DriveInfo is now compatible with Clarion Desktop.
Posted Monday, November 27, 2006

MapPoint Templates

WC Software Development Inc. has released the Clarion for MapPoint Templates, which allow Clarion developers to easily add MS MapPoint 2004/2006 to their applications. Features include: Create and Destroy Pushpin sets with balloon; Create and Destroy Pushpin and Assign them to datasets; Use all of Mappoint internal pushpins; Load internal Pushpin category list as use as your own; Ability to Add Comments and Data to pushpins; Address validation, parsing of address; Geocoding to address (Latitude,Longitude); Geocoding to mouse pointer; Zoom In / Out Control; Altitude control spin box; Add Notes to map - (Will be date and time controlled Next revision); Waypoint (Routing /Directions) - support from dual addresses to many; Calculation support for mileage calculations to include expenses. Demo available. Beta Price is $279.00, once released it will be $389.00.
Posted Monday, November 27, 2006

CoolFrames Beta 1.0

Now in its first beta release, CoolFrames provides custom graphics for your application window frame, caption text and buttons.
Posted Monday, November 27, 2006

1stIconDesign Sale

All 1stIconDesign icon collections are now on sale. You can also save $20 by pre-ordering the People collection.
Posted Monday, November 27, 2006

Gitano Christmas Special

Numerous discounts and special offers are now available on Gitano products.
Posted Monday, November 27, 2006

J-Fax, J-Skype, and J-Spell Bundle

For a limited time you can now buy a bundle consisting of J-Fax, J-Skype, and J-Spell. Combined, these three products are worth $277, but the bundle will only set you back $259. Buy it from the online store at www.clariondesktop.com, you'll get the usual 5% discount, and because these are StrategyOnline products you'll get another 5%. So with 10% off that comes to the low, low price of just $233.10, that's a savings of $43.90. J-Skype will be going gold within the next two weeks, at which time it's price will be increasing, so in effect you'll be saving well over $50 if you take advantage of this offer.
Posted Thursday, November 23, 2006

One Week Left In Capesoft Amnesty

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 23, 2006

Class Anatomy 1.3.0

Class Anatomy 1.3.0 is now available. This release benefits from the bug fix to DriveInfo.

Class Anatomy is also now Clarion Desktop compatible.
Posted Thursday, November 23, 2006

DriveInfoClass Bug Fix

Roel Abspoel has released a new version of the DriveInfo class. This fixes a bug which could result in an endless loop when using EnumerateDrives.
Posted Thursday, November 23, 2006

J-Spell Adds Portuguese

J-Spell 1.33 includes options for Portuguese and Portuguese (Brazilian) in the global extension template.
Posted Thursday, November 23, 2006

Clarion Desktop 2.05

Clarion Desktop 2.05 is now available for download. Changes include: A new password manager with support for multi-line password data; Subscriptions via credit card, PayPal, or ClarionShop; Changes to advertising rates; Various improvements.
Posted Thursday, November 23, 2006

GTL 6.26 Released

Version 6.26 of the Go To Lunch batch compiler introduces a second command line parameter which will terminate GTL when the processing selected in the first parameter completes.
Posted Thursday, November 23, 2006

Clarion Desktop v1 Expires

As Clarion Desktop 2.0 is out, version 1 will stop working this week. Standard (free) and Professional (subscription) versions of Clarion Desktop 2.0 are available.
Posted Thursday, November 23, 2006

Huenuleufu Moves To .COM Domain

Huenuleufu has moved to a new server and has dropped the .ar counry name extension. From now on please go to www.huenuleufu.com for all Huenuleufu products.

Posted Thursday, November 23, 2006

.MOBI Domains At Oak Park Solutions

The .MOBI domain has recently been opened and is proving very popular. To make registering .MOBI domain names easier for Clarion developers and their clients Oak Park is running a sale on .MOBI registrations for a limited time of $14.99/year (2-year min). Also, for those unsure of how to make .MOBI compliant web sites, Oak Park is also running a sale on the WebSite Tonight line of products, starting at $4.99 per month. These products include .MOBI templates that make setting up .MOBI compatible web sites easy.
Posted Thursday, November 23, 2006

Clarion Desktop 2.00 (Gold)

Clarion Desktop 2.00 is the first "Gold" release of Clarion Desktop. It is also the first version to introduce a subscription model. There are two versions, "Standard" and "Professional". Standard Edition remains 100% free, and most of the functionality that you have been using over the past couple of weeks is still available in the free version. Professional Edition is identical to Standard Edition except for additional features. Version 2 also offers suppliers some exciting new features. If you login to ClarionDesktop.com you will see several new fields in the products table, including "Product Description", "Clarion Compatibility Info", "BuyNow URL", and "Price". To add your products directly into the ClarionDesktop.com store contact Gary. ClarionDesktop.com will take 15% commission on any sales, which exactly covers eSellerate and subscriber costs.
Posted Thursday, November 23, 2006

EasyOpenOffice 1.03

EasyOpenOffice 1.03 is now available. This release has numerous new methods as well as several new templates. Requires Clarion 5.5 or Clarion 6.1/6.2/6.3; ABC, Legacy class template support; 32-bits only. Price: $149 (1 Developer license).
Posted Thursday, November 23, 2006

NeatMessage 2.00 Beta

For current users of NeatMessage 1.x or Audit Pack, the NeatMessage 2.00 beta is a free download. You will need a new maintenance code for NeatMessage, available by email request. Numerous new features and changes.
Posted Thursday, November 23, 2006

Dr. Watson Analyzer

This utility analyses DrWatson logfiles and shows(Besides the raw-data) statistics about the found crashes. Analyses can be saved as HTML with the raw-data, HTML with the raw-data in CSV or as XML. Stats include: Percentage of crashes per application; Percentage of crashes per date; Percentage of crashes per user; Percentage of crashes per error.
Posted Thursday, November 23, 2006

SimSoft Clarion Desktop Compatibility

The following templates are now fully compatible with Clarion Desktop: Simsoft Templates; Simshape Templates; Simpad Templates; SimPageOfPage Templates; SimTabTree Templates; SimDatesClass Templates; SimFileLauncher. If you are using Clarion Desktop please re-install and the necessary ini file will now be written.
Posted Thursday, November 23, 2006

Firebird 2.0 Released

Firebird 2.0 is now available for download. Changes include: Improved performance in searching and matching; Removal of 252-byte limit on index size and the 30 Gb limit on table size; New interface for international character sets; 64 bit support; Better security; Support for derived tables; EXECUTE BLOCK syntax; Explicit cursors in PSQL; Optional WAIT lockout conflict timeout; Incremental backup; Serverless local connection protocol on Windows; Complete Services API implementation.
Posted Thursday, November 23, 2006

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

Clarion Magazine

Clarion 101: Understanding Keys and Indexes

by Bruce Johnson

Published 2006-11-16

Organizing your data is important, but data is only valuable if you can quickly and efficiently retrieve it. In the (good) old days good filing clerks were hard to get, and well looked after. Their job was to file away documents in such a way that they could be retrieved later on.

Sounds easy? It’s not. How can you easily reconcile a customer’s account if everything is filed by date? How can you find an invoice where you know the date, and amount, but not the customer?

The problem with paper systems is that they can be sorted only in one specific way. But with computer data the same data can be stored in multiple different ways. Of course the data isn’t stored multiple times, what is needed are multiple indexes into the data.

An index allows the computer to efficiently retrieve a subset of the table. For example, if the table is sorted by last name, then finding all the Johnsons is efficient. The computer can jump to the first Johnson and keep reading the data until the Joikes are encountered.

On the other hand the lack of an index means that records may need to be read and then discarded. For example getting a list of all the people born in the month of May (when there is no index) would require that all the records are read and those with a birth-date in one of the other eleven months discarded. The disk is usually the slowest part of any computer (by several orders of magnitude) so this would be a very inefficient retrieval.

Terminology

In the Clarion world an index into the data is called a key. There are different kinds of indexes (called both indexes and keys) but the differences between them will be discussed later.

In the non-Clarion world, and especially if you deal with SQL databases, indexes are often just called indexes. Technically a key and an index mean slightly different things in SQL, but that’s not important at this point.

Keys

Keys are made up of one or more fields out of the table. For example, in the previous article, there was a Person table that looked something like this:

Person

PersonId	**FirstName**	**LastName**	**Sex**	**Status**
1	Bruce	Johnson	1	2
2	David	Harms	1	2
3	Robert	Zaunere	1	2

This table could easily require sorting in a number of different ways. Sorting by `PersonId`, `FirstName` and `LastName` would all be common sort orders for this kind of table.

Key Conmponents

The key for this table might consist of something like this:

Key Name	**Components**
IdKey	PersonId

FirstNameKey	FirstName,LastName
LastNameKey	LastName,FirstName

As you can see each key has a (unique) name and at least one *component*. They can also have more than one component.

Since each person has a unique `PersonId`, there's no point in having any more components for the `IdKey`. However it's possible that many people will have the same first name, or indeed the same last name, so further sorting by using more components is necessary.

In the `FirstNameKey` the people are sorted by their first name. All the Bruces are together, all the Davids are together and so on. Then the records are further sorted by their `LastName`. So Bruce Barrington would come before Bruce Johnson, and David Bayliss would come before David Harms.

In addition each component can be set to sort in an Ascending or Descending sort order. If there were a `BirthDate` field added to the `FirstNameKey` then all the Bruce Johnsons could be sorted either oldest first, or youngest first.

Key Properties

Apart from the components each key can also have a number of properties. This describes how the key will behave later on when the program uses the table.

The key can be set as *unique*. In the example above the `IdKey` would definitely be set as unique since each person requires a unique `PersonId`. However the `FirstNameKey` should not be unique because a `FirstName`, `LastName` combination does not necessarily result in a unique identifier. There are many, many, Bruce Johnson's in the world. Indexes and Dynamic Indexes (both forms of the seldom-used `INDEX` structure – see below) cannot be set as unique.

Another property is whether the component is c*ase sensitive* or not. Since the `IdKey` only contains a number this isn't important (and indeed makes no sense) but case sensitivity would dramatically alter the `LastNameKey`. If the key was set as case sensitive then Johan de Klerk would come well after Robert Zaunere. In most cases case sensitivity is turned off.

A third common property for a key is a*uto numbering*. If a key is set to auto-number then the last component in the key should be a number, and the key should be set as unique. The system will automatically increment this number for every record that gets added to your system. In the `Person` table, the `IdKey` would probably be set as auto-numbered.

The last common property for keys is to e*xclude empty keys*. This means that records, where all the components in the key are zero (for numbers), or blank (for strings), will not be included in the key at all. So if `IdKey` is set to exclude empty keys, then all the people with a `PersonId` of 0 will not be included in the key, but they will be included in the table. That would not be a good thing because multiple people could have the same ID (0) when everyone should have a unique ID

In some cases excluded empty keys can be very handy. By excluding all the empty records you can have a key that sorts a subset of the data.

Primary Key

All tables should have at least one unique key which is designated as the p*rimary key*. This key should (ideally) have a single component, but more than one component is allowed. Primary keys are, by definition, unique.

While most flat-file systems (such as TopSpeed) do not enforce the requirement of a primary key, most SQL systems will not work well (or at all) without one.

The "hidden" ID field, which I encouraged you to add to all tables, and never show to the end user, makes the perfect component for a primary key. Having this field in the first place, and using it as the primary key, is a good design habit and one you will not regret. If this key is also set to be auto-numbered then you won’t have to worry about specifying unique values at data entry time.

Index types

There are three kinds of index (small i) which Clarion supports.

The first and by far the most common is a `KEY`. This is an index which is maintained all the time for you. If you add, edit, or delete a record, then all the `KEY`s are updated at the same time.

The second, and seldom used, are `INDEX`es (big I). `INDEX`es are predefined sort orders, like `KEY`s, but they are not maintained unless a `BUILD` statement is executed. So there’s no overhead when adding, editing or deleting records. The primary advantages of Indexes though are made somewhat obsolete by fast hard drives and fast computers.

The third option, and one almost never seen, is the dynamic `INDEX`. This is the same as an `INDEX` except that the sort order is defined at run-time, and not pre-defined in the dictionary. This type is supported by most of the flat-file drivers, at least in some form, and is only really useful in some very specific instances (none of which you are likely to encounter.)

In the Clarion world whenever people talk about indexes, they are usually (almost always) referring to keys. When you talk to non-Clarion programmers their "index" is analogous to the Clarion "KEY".

Real World Tip

Do not use national identity numbers as the primary index. While at first glance these should be perfect identifiers there are all too often people who have no number at all. For example recent immigrants to the US do not have a Social Security number. Using the Social Security number as your unique person identifier falls over the moment a visitor arrives at the door.

Plus at the time of capturing the original data, the Social Security number may not be known. Using it as the Id number in the system means that no data can be entered until this one item can be determined.

In addition many of these numbers are, at least nominally, secret. If you don’t need them, don’t store them.

If sort orders are so important why not just create a key for every field in the table? Well nothing comes without a cost. Every time records are added, edited or deleted, all the existing keys need to be updated. This takes extra time. Adding keys may speed up reading the data, but it also slows down writing the data. Finding the correct balance is part of the programmer’s job.

Summary

- Indexes are important in allowing the data to be efficiently retrieved.
- Indexes in Clarion are commonly called *keys*
- Keys can consist of one or more *components*
- Keys have properties such as *primary*, *case sensitive*, *auto numbered*, and *exclude empty*.
- All tables should have a primary key.
- Keys speed up reading data, but slow down the speed with which data can be written into the database.
- Having a single component, primary key, which contains a value unrelated to the record is a design recommendation.

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

- » Whats a "Joike" ?
- » A last name. Bruce would, of course, never use that as a...

Clarion Magazine

Printing A Tree From A Page Loaded Browse

by David Podger and Deon Canyon

Published 2006-11-30

During October and November of 2005, Clarion Magazine published our article A Tree in a Page Loaded Browse: the Sequel, in three parts. Its content built upon two articles by Ronald van Raaphorst which appeared in Clarion Magazine in May 2003.

Taken together, the above articles explained a simple way to implement an "any-level, any-length" tree in a single .tps file and to view and manipulate it from within a standard Clarion page-loaded browse.

There are many uses for such a file structure, but our interest was to use it to represent interactive dialogue. A stripped-down example of a dialogue tree is shown in Figure 1.

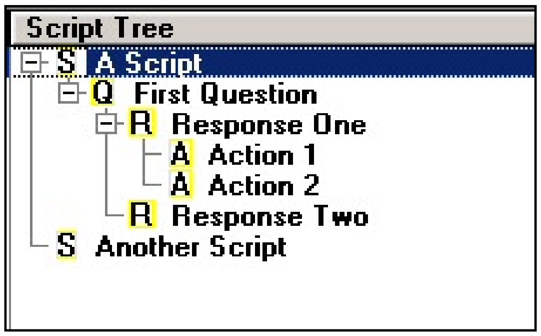


Figure 1. A simple dialogue tree

There are four record types shown in Figure 1:

- **S** Script a script consists of a set of "questions"
- **Q** Question one or more possible responses are attached to each question
- **R** Response "actions" are executed when a given response is chosen
- **A** Action an action executes an expression and saves the result

As can be seen, this tiny tree has a length of seven items, with four levels. In practice our dialogues need no more than 50 levels, but benefit from having no length limit.

At run-time, when a script is executed, the end-user chooses a pathway through the script by selecting from amongst the offered responses. For now, however, we are just interested in the design environment.

Follow-on questions may be attached to responses, as may be seen in Figure 2.

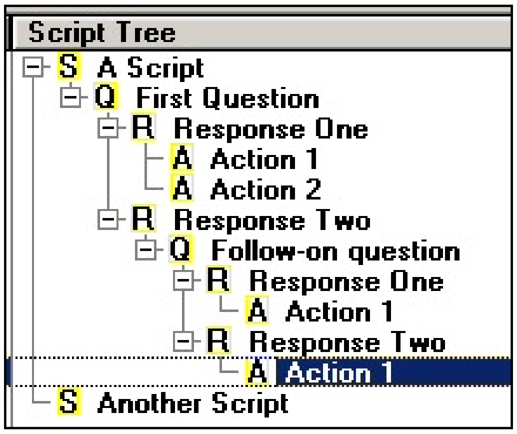


Figure 2. Added follow-on questions

Even more questions (and their associated responses and actions) may be added, to the required depth. It will help at this point to explain Ronald van Raaphorst's contribution to the design of a tree file, using a brief excerpt from our earlier article. He proposed that for each entry in the above tree there should be a hidden `CSTRING` field that controlled everything. In the code below we use his name (`SeqNo`) for this field. It is the only additional data field needed to produce the above tree in a standard page loaded browse.

Here are the control strings for the entries in Figure 2:

Type	Control string	Level
S	0001	1
Q	0001.0001	2
R	0001.0001.0001	3
A	0001.0001.0001.0001	4
A	0001.0001.0001.0002	4
R	0001.0001.0002	3
Q	0001.0001.0002.0001	4
R	0001.0001.0002.0001.0001	5
A	0001.0001.0002.0001.0001.0001	6
R	0001.0001.0002.0001.0002	5
A	0001.0001.0002.0001.0002.0001	6
S	0002	1

Each additional level in the tree adds another five characters to the control string.

The tree file is kept in the sort order dictated by this string. A brief inspection of the above control strings will confirm that they are indeed in that order. The decimal points that separate each level are for readability only, but very handy for that, as can be seen. When it comes to inserting additional entries anywhere in the tree, there is no problem, there is room for thousands of entries.

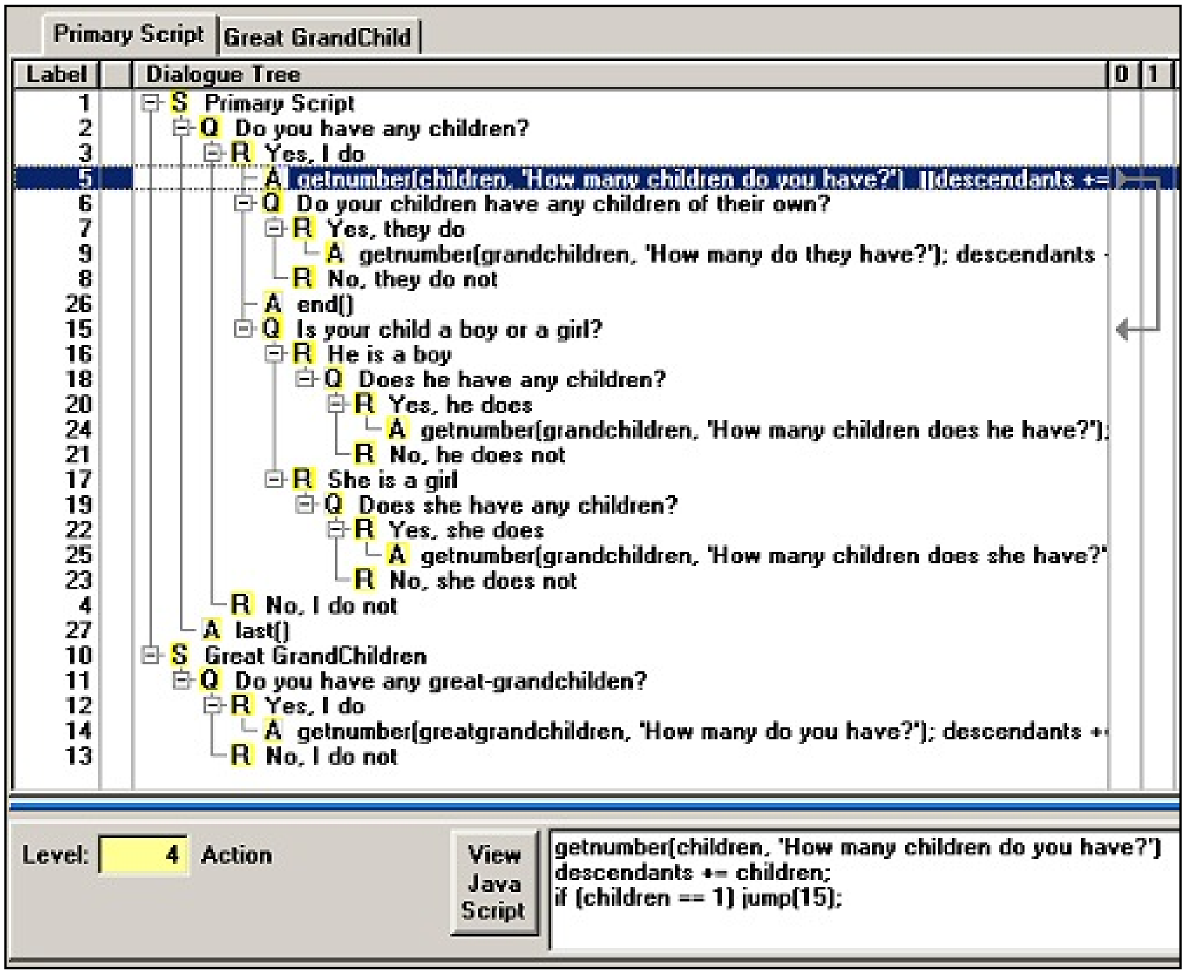
What about the Level values shown in the above table: where do they come from? As explained, the control string provides everything. It gives the level number, by using this simple expression: `Level = INT(LEN(SeqNo) / 5) + 1`.

As will be seen below, the code that prints a dialogue tree does so by re-generating its structure from the `SeqNo` field and from nothing else.

In the time that has passed since the above article was written we have been attempting some longish case studies and eventually ran into the need for hard copy. The output needed is more like a flowchart or diagram, because dialogue trees can splay out sideways and run to a number of pages. Once on paper they can be studied, shared with a group and revised before being modified within the development environment. Creating a good dialogue is a non-trivial task.

Mercifully, a kind of inevitable discipline exerts itself on the authors of interactive dialogues. If they entertain too many possibilities they find their dialogue has become unmanageable. Typically, a "green" line of connected responses is defined, which runs through the dialogue from beginning to end. Then there are "orange" lines that run for a while, but ultimately fizzle out or rejoin the main stream. "Red" lines are more swiftly curtailed.

In a complex dialogue there may of course be more than one green line, but normally there are only a few. The effect of this discipline, imposed as it were by common sense, is that dialogue trees do not splay out too far. They take on a vertical orientation, generally work their way downwards and do not sprawl unpredictably. This simple tendency informed our solution. Figure 3 shows a brief example dialogue. It asks a few questions to determine how many descendants the respondent has. It has a depth of eight levels.



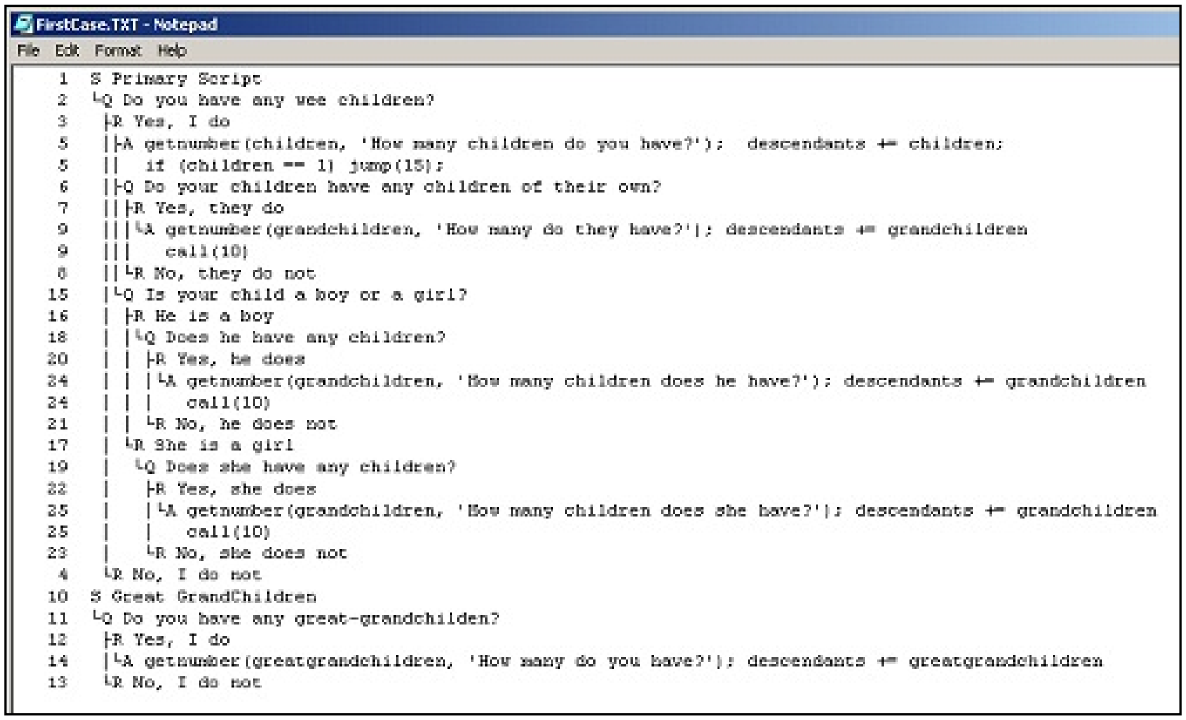
```
Label   Dialogue Tree                                                                   0 1
  1     S Primary Script
  2       Q Do you have any children?
  3         R Yes, I do
  5           A getnumber(children, 'How many children do you have?'); [descendants +=
  6           Q Do your children have any children of their own?
  7             R Yes, they do
  9               A getnumber(grandchildren, 'How many do they have?'); descendants
  8             R No, they do not
 26           A end()
 15           Q Is your child a boy or a girl?
 16             R He is a boy
 18               Q Does he have any children?
 20                 R Yes, he does
 24                   A getnumber(grandchildren, 'How many children does he have?');
 21                 R No, he does not
 17             R She is a girl
 19               Q Does she have any children?
 22                 R Yes, she does
 25                   A getnumber(grandchildren, 'How many children does she have?'
 23                 R No, she does not
  4         R No, I do not
 27       A last()
 10     S Great GrandChildren
 11       Q Do you have any great-grandchildren?
 12         R Yes, I do
 14           A getnumber(greatgrandchildren, 'How many do you have?'); descendants +
 13         R No, I do not

Level: 4  Action

getnumber(children, 'How many children do you have?')
descendants += children;
if (children == 1) jump(15);
```

Figure 3. An example dialogue (view full size image)

The browse in Figure 3 is a standard Clarion browse of a tree. Please refer to the earlier articles to see how it works. The full content of the highlighted entry in the browse is shown in a hot field below the browse. The pseudo code in that hot field shows the simple downward flow of a dialogue being bypassed using a *jump* function (logically identical to a goto). Jumps are represented diagrammatically to the right of the browse and may go forward or back.

So, how could the tree in Figure 3 get onto hard copy? One way would be to use third-party tools that allow printing direct from a browse. Used to the full, these tools will print everything in the tree, including the line icons that delineate the tree and the characters "**S**", "**Q**", "**R**", and "**A**", which are also icons. But they are a resource-heavy way to go for just one output, when you add up all the tools needed and their impact on program size.

The solution illustrated in Figure 4 presents the tree as a standard ASCII file. Pretty it is not. But being in this unadorned format it can be viewed by Notepad and printed.



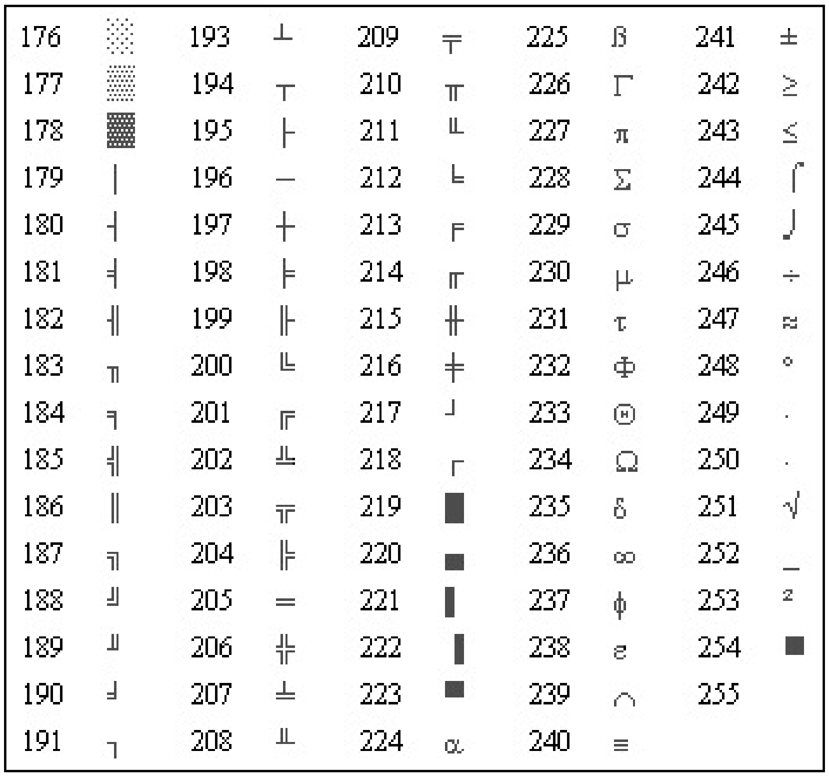
```
  1   S Primary Script
  2   └Q Do you have any wee children?
  3    ├R Yes, I do
  5    │├A getnumber(children, 'How many children do you have?');  descendants += children;
  5    ││  if (children == 1) jump(15);
  6    │├Q Do your children have any children of their own?
  7    ││├R Yes, they do
  9    │││└A getnumber(grandchildren, 'How many do they have?'); descendants += grandchildren
  9    │││    call(10)
  8    ││└R No, they do not
 15    │└Q Is your child a boy or a girl?
 16    │ ├R He is a boy
 18    │ │└Q Does he have any children?
 20    │ │ ├R Yes, he does
 24    │ │ │└A getnumber(grandchildren, 'How many children does he have?'); descendants += grandchildren
 24    │ │ │    call(10)
 21    │ │ └R No, he does not
 17    │ └R She is a girl
 19    │  └Q Does she have any children?
 22    │   ├R Yes, she does
 25    │   │└A getnumber(grandchildren, 'How many children does she have?'); descendants += grandchildren
 25    │   │    call(10)
 23    │   └R No, she does not
  4    └R No, I do not
 10   S Great GrandChildren
 11   └Q Do you have any great-grandchilden?
 12    ├R Yes, I do
 14    │└A getnumber(greatgrandchildren, 'How many do you have?'); descendants += greatgrandchildren
 13    └R No, I do not
```

Figure 4. A plain solution (view full size image)

Note that the expressions in an Action entry can be as long as required, since they may be spread over multiple lines. This article does not cover the printing of jump lines, which are a bit specific to a dialogue, but it does explain the code needed to do the tree itself.

Getting started on coding

The line characters in Figure 4 are to be found in a font called MS Linedraw. Its file name is LineDraw.ttf and it can be downloaded and made available to any Clarion program, or be selected from within Notepad. Google lists a number of download sites. The line characters are identical to those in the old DOS extended character set and have the same decimal values. A Google search on "ASCII codes" provides convenient tables. Figure 5 shows an extract from one such table. Each character's decimal value and appearance is shown.

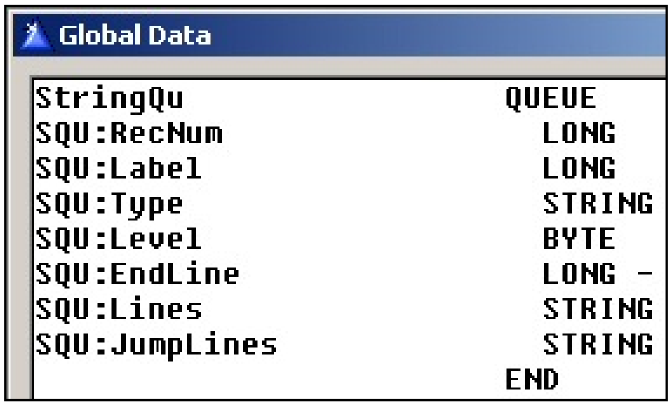


176	░	193	┴	209	╤	225	ß	241	±
177	▒	194	┬	210	╥	226	Γ	242	≥
178	▓	195	├	211	╙	227	π	243	≤
179	│	196	─	212	╘	228	Σ	244	⌠
180	┤	197	┼	213	╒	229	σ	245	⌡
181	╡	198	╞	214	╓	230	µ	246	÷
182	╢	199	╟	215	╫	231	τ	247	≈
183	╖	200	╚	216	╪	232	Φ	248	°
184	╕	201	╔	217	┘	233	Θ	249	∙
185	╣	202	╩	218	┌	234	Ω	250	·
186	║	203	╦	219	█	235	δ	251	√
187	╗	204	╠	220	▄	236	∞	252	ⁿ
188	╝	205	═	221	▌	237	φ	253	²
189	╜	206	╬	222	▐	238	ε	254	■
190	╛	207	╧	223	▀	239	∩	255	
191	┐	208	╨	224	α	240	≡		

Figure 5. An extract of extended ASCII codes

Some preliminary manipulation of the `TRE:SeqNo` field in the `TREE` file is needed to create a tree ready for printing. This is done in three stages using a queue called `StringQu`. The queue is defined globally so as to be accessed by multiple procedures.

The fields in `StringQu` are shown in Figure 6, an extract from the data definition entered into the Global Properties of the app.



```
StringQu                 QUEUE
SQU:RecNum                 LONG
SQU:Label                  LONG
SQU:Type                   STRING
SQU:Level                  BYTE
SQU:EndLine                LONG -
SQU:Lines                  STRING
SQU:JumpLines              STRING
                         END
```

Figure 6. The fields in StringQu

The queue fields are:

`SQU:RecNum`	Holds running number from 1 to the number of entries in the queue

SQU:Label	Holds the contents of the TRE:TreeAuto field, an unchanging number identifying every record in the TREE file.
SQU:Type	A single character field containing "**S**", "**Q**", "**R**" or "**A**"
SQU:Level	The level value calculated from the TRE:SeqNo field
SQU:EndLine	The TRE:TreeAuto value of the record marking the end of the vertical line which began at the TRE:TreeAuto in SQU:Label
SQU:Lines	A 50 character field holding the tree lines as they will be printed
SQU:JumpLines	Not used in this article

The only fields that are read from the TREE file are:

TRE:PlayAuto	an unchanging number identifying each "Play" or Case Study
TRE:SeqNo	the hidden CSTRING field that defines a tree's structure
TRE:TreeType	a single character field containing "**S**", "**Q**", "**R**" or "**A**"
TRE:TreeAuto	an unchanging number identifying every record in the TREE file

Note that the queue entries are of a manageable size, principally because they do not need to hold the contents of the TRE:SeqNo field.

The routine doing the work is CalcLines and its job is to produce a finished StringQu which can then be used to create an ASCII file in the format shown in Figure 4. The main task of the first of the three stages is to extract needed fields out of TREE and write them to StringQu as follows:

```
CalcLines       ROUTINE
  ! build initial version of StringQu
  FREE(StringQu)
  CLEAR(TRE:Record)
  STREAM(TREE)
  ! confine to one Case Study
  TRE:PlayAuto = GLO:Play
  ! key has two fields: TRE:PlayAuto and TRE:TreeAuto
  SET(TRE:Key_SeqNo,TRE:Key_SeqNo)
  LOOP
    NEXT(TREE)
    IF ERRORCODE() THEN BREAK.
    IF TRE:PlayAuto <> GLO:Play
      ! out at end of the one Case Study
      BREAK
    End
        ! Script, Question, Response, Action
    SQU:Type = TRE:TreeType[1]
```

```
            ! save unchanging ID of TREE record
          SQU:Label = TRE:TreeAuto
          IF SQU:Type[1] = 'S'
                ! just gets the indenting right
            SQU:Lines[1] = ' S'
          ELSE
                ! target for lines during 3rd pass
            SQU:Lines = ''
          END
          SQU:Level = INT(LEN(TRE:SeqNo) / 5) + 1
              ! calculated in the next stage
          SQU:EndLine = 0
          ADD(StringQu)
        END
        FLUSH(TREE)
```

The above code pretty much explains itself, with some help from the comments. At its conclusion all the information needed to construct a printable tree is contained in `StringQu`, but it needs some further work.

The second stage of the `CalcLines` routine reads `StringQu` for just one purpose – to calculate a value for `SQU:EndLine`. When this is done the level, starting point and length of every vertical line in the tree will be known. The trick is to read the queue backwards, saving up line end points until they can be written back into the entries where the respective lines began.

```
        ! calculate ending RecNum for each line and
        ! save in the record where the line begins
        PrevLevel = 0
        ! array of 50 LONG's, one per level
        CLEAR(LevelArray)
        ! read queue backwards
        LOOP RecNum = RECORDS(StringQu) TO 1 BY -1
          GET(StringQu,RecNum)
          IF SQU:Level < PrevLevel
                ! if going up to a higher
                ! level in tree then write
                ! array value at index

            SQU:EndLine = LevelArray[SQU:Level+1]
            ! SQU:Level+1 to SQU record
                ! and zero entry in array
            LevelArray[SQU:Level+1] = 0
          END
              ! save RecNum for tracing
          SQU:RecNum = RecNum
              ! purposes only
          PUT(StringQu)
              ! if level has changed and no
              ! value is present as yet, then
              ! save RecNum in the SQU:Level'th
              ! position in LevelArray
          IF SQU:Level <> PrevLevel
            IF ~LevelArray[SQU:Level]
              LevelArray[SQU:Level] = RecNum
            END
          END
              ! and update PrevLevel
          PrevLevel = SQU:Level
        END
```

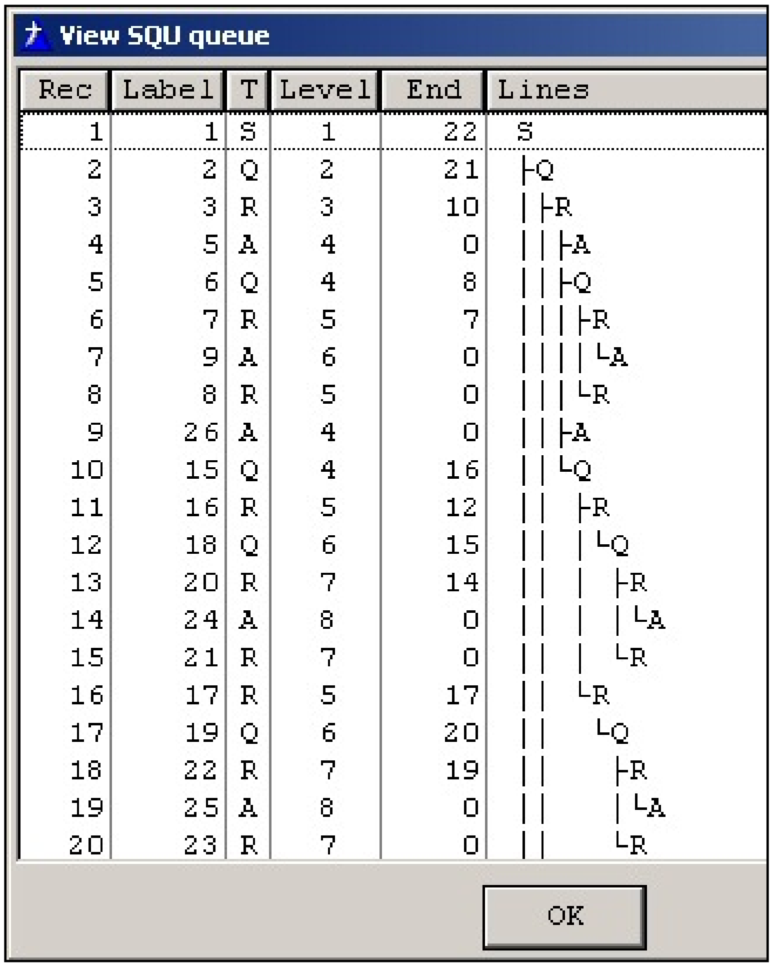
For those who are interested, the best way to understand the above code is not for me to attempt an essay on it! Rather, study the comments as you work through a pencil and paper exercise and see that it works.

In the last stage the `LevelArray` is cleared and used again. It is here that the `SQU:Lines` field can be filled in with the special line drawing characters. All the lines across one entry are generated by the inner `LOOP` based on what is found in the `LevelArray`.

```
      ! now, drop in the line characters
      CLEAR(LevelArray)
       ! read forwards in StringQu
      LOOP RecNum = 1 TO RECORDS(StringQu)
        IF RecNum = 1
          GET(StringQu,RecNum)
              ! hold for next entry in queue
          NextNum = SQU:EndLine
          SQU:Lines = ' S'
          PUT(StringQu)
        ELSE
          GET(StringQu,RecNum)
          IF NextNum >= RecNum
                ! if Endline is equal or less
                    ! than current record then the
                    ! SQU:EndLine from the
                    ! preceding record was written to.
            LevelArray[SQU:Level] = NextNum
          END
          LOOP I# = 1 TO SQU:Level
                ! writes lines to each position
                    ! in SQU:Lines up to SQU:Level
                    ! but only if LevelArray contains
                    ! an EndLine value there
            IF LevelArray[I#]
                      ! if so, decide which line char:
              IF I# < SQU:Level
                        ! a vertical line
                SQU:Lines[I#] = '<179>'
              ELSIF LevelArray[SQU:Level] = RecNum
```

```
                    ! an ending corner
            SQU:Lines[I#] = '<192>'
            SQU:Lines[I#+1] = SQU:Type
            LevelArray[I#] = 0
          ELSE
                    ! or mid-line connector
            SQU:Lines[I#] = '<195>'
            SQU:Lines[I#+1] = SQU:Type
          END
          PUT(StringQu)
        END
      END
      IF SQU:Level = 1
        SQU:Lines = ' S'
        PUT(StringQu)
      END
          ! hold for next entry in queue
      NextNum = SQU:EndLine
    END
  END
```

For tracing purposes `StringQu` can be viewed in a `List` control, just to be sure it is okay. An example is shown in Figure 7.



View SQU queue

Rec	Label	T	Level	End	Lines
1	1	S	1	22	S
2	2	Q	2	21	├Q
3	3	R	3	10	│├R
4	5	A	4	0	││├A
5	6	Q	4	8	││├Q
6	7	R	5	7	│││├R
7	9	A	6	0	││││└A
8	8	R	5	0	│││└R
9	26	A	4	0	││├A
10	15	Q	4	16	││└Q
11	16	R	5	12	││ ├R
12	18	Q	6	15	││ │└Q
13	20	R	7	14	││ │ ├R
14	24	A	8	0	││ │ │└A
15	21	R	7	0	││ │ └R
16	17	R	5	17	││ └R
17	19	Q	6	20	││ └Q
18	22	R	7	19	││ ├R
19	25	A	8	0	││ │└A
20	23	R	7	0	││ └R

Figure 7. Displaying StringQu in a List

At this stage in the proceedings, the structure of the dialogue tree has been extracted successfully. The next steps is to join it up with the tree's content, take care of Action entries that will run for more than one print line, and output an ASCII file. These tasks are addressed in the routine `BuildTREEFILE`.

The `TREEFILE` is defined in the After File Declarations Global embed:

```
TreeName STRING(64)
TREEFILE FILE,DRIVER('ASCII'),NAME(TreeName),|
           PRE(TRA),CREATE
 Record    RECORD,PRE()
 Line        STRING(255)
           END
         END
```

The routine begins by using the Case Study name to give a name to the ASCII file. It then proceeds to `CREATE` the file before it is opened, which has the effect of deleting any existing file of that name and beginning with an empty ASCII file..

```
BuildTREEFILE      ROUTINE
  ! form the name for the TREEFILE to be
  ! generated, using the Case name
  TreeName = CLIP(PATH()) & '\Generate\' & |
    CLIP(LOC:CaseName) & '.TXT'
  CREATE(TREEFILE)
  IF ERRORCODE()
    MESSAGE('Failed creating TREEFILE')
  END
  OPEN(TREEFILE)
  IF ERRORCODE()
    MESSAGE('Failed opening TREEFILE')
  END
```

The routine then reads all the records in the `TREE` file for the case and reads each matching entry from `StringQu`.

```
  T# = 0
  CLEAR(TRE:Record)
  STREAM(TREE)
  TRE:PlayAuto = GLO:Play
  SET(TRE:Key_SeqNo,TRE:Key_SeqNo)
  LOOP
    NEXT(TREE)
    IF ERRORCODE() THEN BREAK.
    IF TRE:PlayAuto <> GLO:Play
      BREAK
    END
```

```
        T# += 1
        GET(StringQu,T#)
```

A matching entry from `StringQu` is read for each `TREE` record, and a record in the ASCII file is built from a fixed length `SQU:Label`, a variable length set of vertical lines from `SQU:Lines` and the text in the `TRE:TreeText` field. As needed, long entries are formed into multiple records.

```
        X# = INSTRING('<13>',TRE:TreeText,1,1) ! divide TreeText if CR there
        IF X#
              ! blank the first CR found
          TRE:TreeText[X#] = ' '
          TRA:Line = FORMAT(SQU:Label,@n_5) & ' ' & CLIP(SQU:Lines) & |
                     ' ' & TRE:TreeText[1 : (X#-1)]
        ELSE
          TRA:Line = FORMAT(SQU:Label,@n_5) & ' ' & CLIP(SQU:Lines) & |
                     ' ' & TRE:TreeText
        END
        ADD(TREEFILE)
        IF ERRORCODE()
          MESSAGE('Failed adding to TREEFILE')
        END
        L# = LEN(CLIP(TRE:TreeText))
            !reduce S# - clip off last line & icon
        S# = LEN(CLIP(SQU:Lines)) - 2
        CLEAR(SaveLines)
            ! these are the lines to be duplicated
            ! in the following rows
        SaveLines = SQU:Lines[1 : S#]
            ! if needed, add extra rows to dialogue
        LOOP WHILE X#
              ! save beginning of remainder of TRE:TreeText
          Y# = X# + 2
              ! see if another CR/LF
          X# = INSTRING('<13>',TRE:TreeText,1,1)
              ! more to chop up?
          IF X#
                ! blank the CR found
            TRE:TreeText[X#] = ' '
                    ! ending corner needs no following vertical
            IF SQU:Lines[S# + 1] = '<192>'
              TRA:Line = FORMAT(SQU:Label,@n_5) & ' ' |
                & CLIP(SaveLines) & ' ' & TRE:TreeText[Y# : (X#-1)]
            ELSIF SQU:Lines[S# + 1] = '<195>'
                     ! but mid-line connector needs one
              TRA:Line = FORMAT(SQU:Label,@n_5) & ' ' |
                & CLIP(SaveLines) & |
                '<179>  ' & TRE:TreeText[Y# : (X#-1)]
            END
          ELSE
```

```
            IF SQU:Lines[S# + 1] = '<192>'
                      ! ending corner needs no following vertical
              TRA:Line = FORMAT(SQU:Label,@n_5) & ' ' |
                & CLIP(SaveLines) & |
                ' ' & TRE:TreeText[Y# : L#]
            ELSIF SQU:Lines[S# + 1] = '<195>'
                       ! but mid-line connector  needs one
              TRA:Line = FORMAT(SQU:Label,@n_5) & ' ' |
                & CLIP(SaveLines) & |
                '<179>  ' & TRE:TreeText[Y# : L#]
            END
              END
          ADD(TREEFILE)
          IF ERRORCODE()
            MESSAGE('Failed adding to TREEFILE')
          .
        END
      END
      FLUSH(TREE)
      CLOSE(TREEFILE)
```

And that is it. The end result is an output working diagram at a very low cost in program size and which is fast and effective. Figure 4 has already shown what it looks like.

David Podger has been an independent Clarion developer in Australia for a decade. He presently lives in Sydney, where he sells a specialised accounting application for remote communities.

Dr Deon Canyon is a Senior Lecturer (translation: Associate Professor) in public health and medical entomology at James Cook University in North Queensland, Australia. His current research interests include: Disease control simulation models for use in distance education; *Lymphatic filariasis* (a disease caused by thread-like parasitic worms) and the *Aedes polynesiensis* mosquito in transmission and control of the Pacific Head Lice (*Pediculus capitis*); Health in remote settlements and the use of touch screen kiosks to improve it; Aedes aegypti mosquito behavior, physiology and ecology . Deon is married, and has three children. His hobby of choice is kite surfing.

Reader Comments

Add a comment

Clarion Magazine

Controlling Controls

by Steven Parker

Published 2006-11-30

I have a customer browse which must indicate whether a customer has an email address, and if the email address is present, allow the user to send an email. But as usually seems to happen, the obvious solutions don't apply.

My customer file has a field for the customer's email address. To determine that a particular customer has an email address, I could populate the email address in the browse. If a customer does have an email address, it would show. If the customer does not, the column would be empty. Unfortunately, as Figure 1 shows, I don't have the room in the list box for another field.

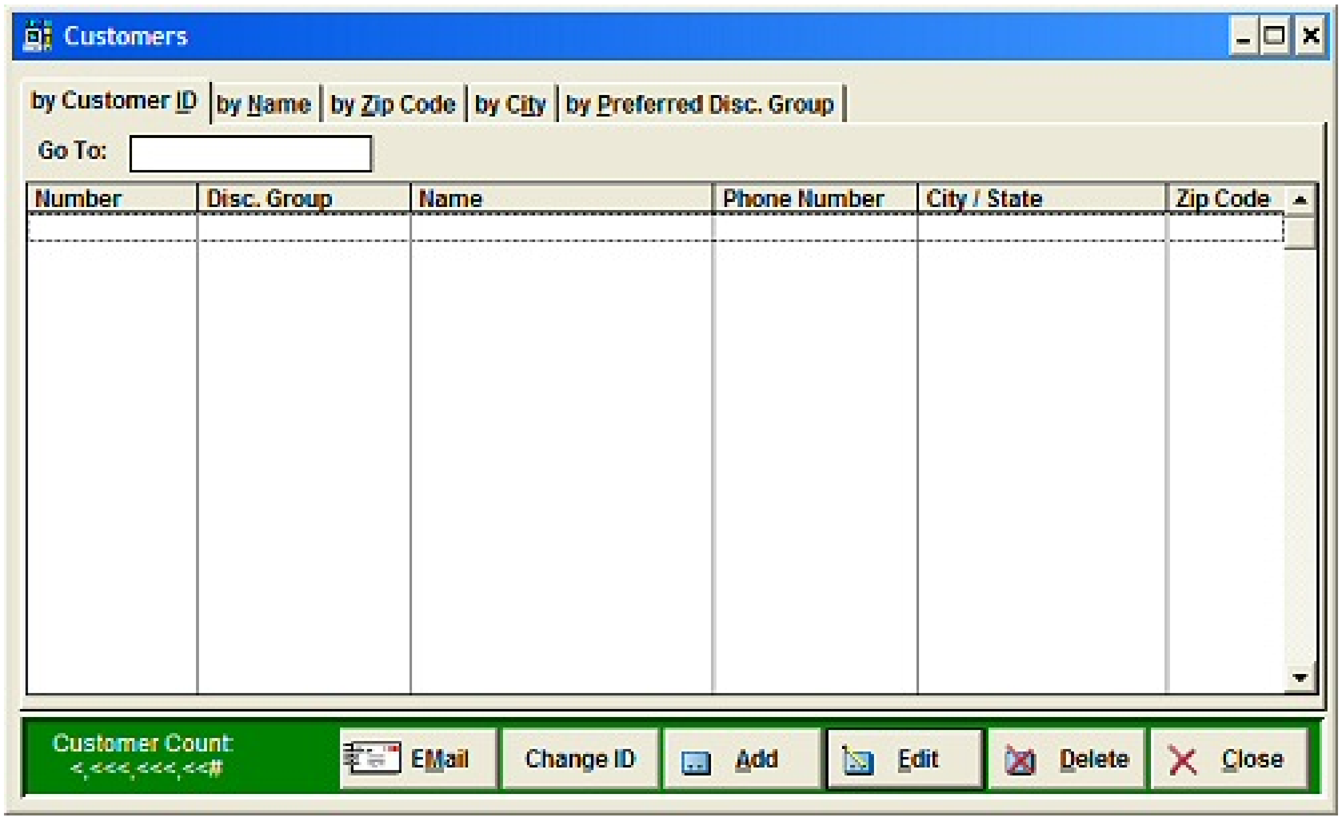


Figure 1. Customer list with "EMail" button (view full size image)

Alternately, I could populate the email field as a hot field, outside the list box proper. The net effect would be substantially the same as having the datum in the list box. Unfortunately, as is the case in so many of my browses, I don't have the real estate for this either.

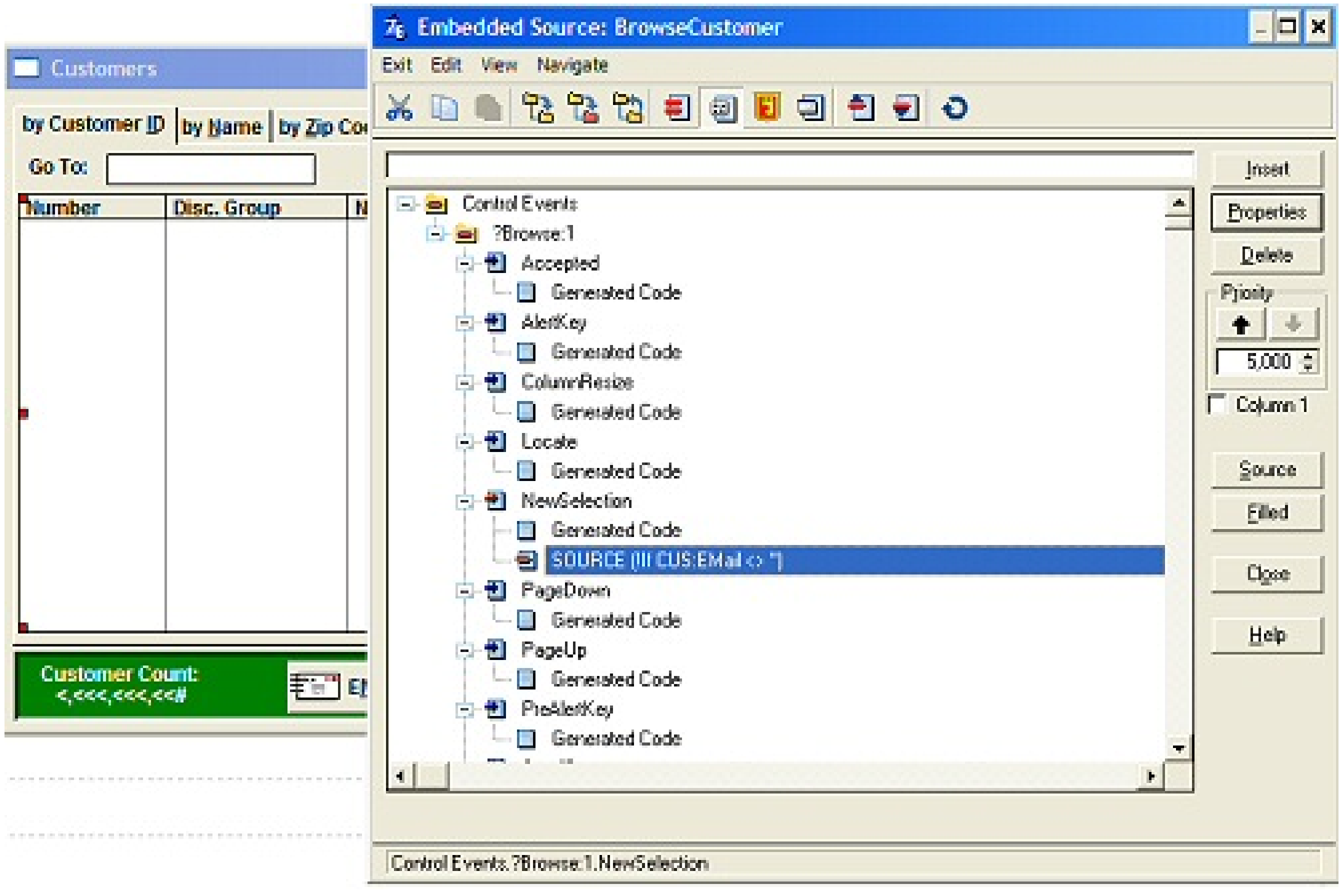
Besides, what I really want to do is click on something to actually send an email. So, I chose a button.

My options, in this case, are quite limited. I need to be able to enable/disable the button as I scroll through this list. This is fortuitous because the button calls my email client with a "send to" value filled in the email address field instead of forcing me to open the form to send an email. As a result I really need the state of the button to reflect the reality of the underlying record.

The question is: "How do I dynamically enable/disable a control on a Browse template procedure?"

A Blast From the Past

In Clarion for DOS, I used the Process Selected Record embed for this purpose. The equivalent embed in windows is `NewSelection`.



```
Control Events
  ?Browse:1
    Accepted
      Generated Code
    AlertKey
      Generated Code
    ColumnResize
      Generated Code
    Locate
      Generated Code
    NewSelection
      Generated Code
      SOURCE (IIf CUS:EMail <> '')
    PageDown
      Generated Code
    PageUp
      Generated Code
    PreAlertKey
      Generated Code
```

Control Events.?Browse:1.NewSelection

Figure 2: NewSelection embed (view full size image)

Note that this method is a method of the list box, not the window.

Suppose I construct a browse where `CUS:Email` is hot and the code in `NewSelection` is:

```
If CUS:EMail <> ''
  ?Email{Prop:Disable} = False
Else
  ?Email{Prop:Disable} = True
End
```

After compiling, I enter some records, some with, some without email addresses. As I scroll through the browse, the email button would indeed enable/disable correctly in response to the underlying data.

However, if I use the mouse to click on a record, page up/down the button no longer updates correctly.

Back to the drawing board

It turns out that TakeNewSelection is the real replacement for the DOS Process Selected Record.

As Arnor Baldvinsson observed in a recent posting:

> TakeNewSelection is executed upon the control [the list control, in the current case] receiving EVENT:Accepted or EVENT:NewSelection. If you put [your code] in EVENT:NewSelection you may also need the put it in EVENT:Accepted. If you use TakeNewSelection you only need one place for the code.

`TakeNewSelection`, as a quick look through the browse class (abbrowse.clw) shows, is called from `BrowseClass.TakeEvent`. This means that any browse event will check whether there has been a new list row selected.

In theory this means that both scrolling up and down the list and mouse clicking on different rows will call this method. And practice proves the theory. In the sample app, downloadable at the end of this article (the app is written in 5.5), select Browse | Browse Customers (New Selection). Create some records, some with, some without email address. Scroll, click, move about the list and you will see that the EMail button enables and disables just as you would expect.

A fly in the ointment

Perhaps you noticed that when you first opened the browse, and there were no customer records, the EMail button was enabled.

Of course, this behavior is incorrect (no list box event has been called, so `TakeNewSelection` isn't either). The obvious solution is to default the button to "disabled." Simply click the "Disable" checkbox in the window formatter for the button. This works and is what I have done in the example application.

In fact, I hadn't really noticed this until Jeff Slarve offered his alternative to

`TakeNewSelection`:

> If you’d look at the updatewindow method, you’d see that’s what it’s for. That’s where the ABC browse does it’s enabling/disabling of the change/delete/whatever buttons.
>
> Also, if there are no records in the browse, your code will not get called. UpdateWindow … will.

The UpdateWindow method

The UpdateWindow method is called by:

- `ResetFromAsk`
- `ResetFromBuffer`
- `TakeAcceptedLocator` (this is where controls are enabled/disabled, e.g. if there are records in the browse)
- `FormVCRClass`

In other words, `UpdateWindow` is called almost as often as `TakeNewSelection`. Importantly, it is the various `Reset` methods that are called at browse initialization, before there are any list box events to handle. This is what prompts Mr. Slarve’s comment that `UpdateWindow` will behave correctly when there are no records without further developer action.

Summary

If you need to dynamically enable/disable update buttons (Insert, Change, Delete), this is the way to do it. The only difference is in the code called:

```
If
  ?BRWx.InsertControl = 0
  ?Insert{Prop:Disable} = True
Else
  ?BRWx.InsertControl = ?Insert
  ?Insert{Prop:Disable} = False
End
```

`TakeNewSelection`, `UpdateWindow`. Both work. Both work as expected. Neither requires much work.

How typically Clarion.

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

How To Use The Debugger, Part 2 (video)

by Richard Rose

Published 2006-11-28

Part 2 of Richard Rose's Camtasia presentation on how to use the debugger is now available to ClarionMag subscribers by special arrangement with the UK Clarion User's Group. This is a 10 minute, 25 meg AVI file. If you have trouble downloading please delete any partial downloads and do a Save As.

Please post any comments below.

Download the video

Reader Comments

Add a comment

- » Dear Richard Thank you for sharing your knowledge about...
- » Hi Braam I've used a product called Camtasia Studio 4...

Clarion Magazine

Solving Problems With Finite State Machines

by John Christ

Published 2006-11-24

Over the years I've done a lot of parsing of text files, some of it trivial and some fairly complex. Searching for text doesn't always mean just finding one string inside another string; often it means finding a string that occurs after another string, or within a specific block of text, or only after a specified string has occurred x number of times. Often times the code winds up looking very spaghetti-like, with lots of flag variables to keep track of where I've been and where I'm going.

Does complex text searching always mean kludgey code? Hopefully not. There's a technique that's been around a long time that I think is quite useful for solving a variety of problems. It's called a Finite State Machine (abbreviated hereafter as FSM). Briefly, you determine the possible states your logic can be in, and then determine what causes transitions between the states. All finite state machines have an initial state, and hopefully a terminal state. All that's left is finding the states in-between.

In this article I'm applying the FSM technique to searching text, but it can be used to solve many coding problems.

An example

Consider the following block of text, which is one of many entries in a file a DHCP server uses to record IP address leases:

Listing 1

```
lease 192.168.0.8 {
  starts 5 2006/10/13 16:39:17;
  ends 5 2006/10/13 22:39:17;
  binding state active;
```

```
    next binding state free;
    hardware ethernet 00:0f:ea:46:98:89;
    uid "\001\000\017\352F\230\211";
    client-hostname "office";
  }
```

Say you want to extract the IP address for a specific client-hostname. "Simple!" you say, "just look for the line that starts with `'lease'` and the line that starts with `'client-hostname'`." Simple, yes, but not terribly rigorous, and it's not guaranteed to work at all. These files use a C-like syntax, and in C, curly brackets group items and semicolons separate them. Newlines tend not to mean much. In fact, the specification for this file just says this about a lease declaration:

```
  lease ip-address { statements... }
```

Standard, line-oriented text searching won't necessarily work in this situation. And in any case I dislike arbitrary line or field length limitations. Unfortunately, Clarion's disk access is primarily record or block oriented. I've always missed the character I/O capability C programmers take for granted.

With these considerations in mind, I've broken the job into three tasks, each building on the one before it:

1. Create a class which performs character I/O, using the DOS driver to retrieve blocks of characters, but encapsulating the ugly details of passing them along one at a time. This will then produce a character stream for the parser.
2. Create a parsing function, which reads characters and creates tokens from the input stream.
3. Finally, create a finite state machine, which will act on the tokens and extract the required information.

The Character I/O Class (CharFileIO)

The C Standard Library provides a variety of functions for performing character I/O, and the `CharFileIO` class mimics this capability. The class only needs Input, not Output, so the functions implemented in the class are:

1. fopen - open a file
2. fclose - close the file
3. getc - get the next character (as a LONG) from the file
4. ungetc – push back one character if we read too far

The class uses the DOS file driver to retrieve up to 512 characters at a time. The buffer length and buffer position are private properties of the class. The getc method:

1. Checks buffer length and position to make sure there are characters in the buffer and that it isn't about to read past the end of the buffer. If it is, the buffer is refilled with the next 512 characters.
2. If there are available characters, returns the next one as an ASCII code. Otherwise `EOF` (-1) is returned.

The `ungetc` function allows you to push back one character. For example, when skipping over whitespace (spaces, tabs, etc.) you don't know you've hit a non-whitespace character until you've read it via `getc`. `Ungetc` allows you to put that one character back at the head of the input stream.

A simple example of using the class is found in Listing 2.

Listing 2.

```
ReadFile PROCEDURE(STRING pTheFile)
fin &CharFileIO
c   LONG
  CODE
  fin &= NEW(CharFileIO)
  fin.fopen(pTheFile)
  LOOP
    c = fin.getc()
    IF c = EOF
      BREAK
    END
    ! Do something with the character
  END
  fin.fclose()
```

FSM1.zip provides a simple demonstration program for the Character I/O class.

The Parsing Function (GetToken)

While the character I/O class is intended to be generic, the parser must be adapted to the grammar of the subject text. In the case of the DHCP server file I need to:

1. Skip over leading non-relevant character, such as comments and whitespace. In this file, comments are lines starting with a '#' and whitespace is tab, linefeed, carriage return and space (see Listing 3.)
2. Decide what characters delimit tokens (See Listing 4.)
3. Decide whether/how to handle quoted strings (See Listing 4.)

Listing 3:

```
LOOP
  c = fin.getc()
  CASE c
  OF EOF
    RETURN NULL
  OF VAL('#') ! comment, scan to end-of-line or EOF
    LOOP
      CASE fin.getc()
      OF 10 ! Linefeed
        BREAK
      OF EOF
        RETURN NULL
      END
    END
  OF    9 ! Tab
  OROF 10 ! Linefeed
  OROF 13 ! Carriage return
  OROF 32 ! Space
    ! Whitespace, keep going
  ELSE
    ! Non-whitespace encountered, read too far.
    ! Push the last character back and exit the loop
    fin.ungetc(c)
    BREAK
  END
END
```

The loop breaks when it encounters non-whitespace, and the next bit of code dynamically creates a token:

```
token &= NEW CSTRING(maxTokenLength)
```

If necessary the size can be increased later. But for now, the code begins copying the input stream to the token (see Listing 4.)

Listing 4.

```
LOOP
  c = fin.getc()
  CASE c
  OF EOF
    BREAK
  OF VAL('"') ! Double quote, toggle the flag and skip the character
    inQuote2 = 1 - inQuote2
    CYCLE
  OF 10   ! Linefeed
```

```
      OROF 13 ! Carriage return
        ! These terminate the token
        BREAK
      OF    9       ! Tab
      OROF 32       ! Space
      OROF VAL(';') ! Semicolon
        ! If inside double quotes, include these characters in the token
        ! Otherwise, these characters terminate the token
        IF inQuote2 = 0
          BREAK
        END
      END
      len += 1
      token[len] = CHR(c)
    END
```

The Finite State Machine

It’s now time to design the state machine, and the first step is to determine what states will be needed. It will probably be helpful to have Listing 1 available as you read this list of states.

- State 0 is the initial state. The code calls `GetToken` until it returns "lease". This is the first element of a lease declaration, which signifies a change of state. Move to state 1.
- State 1 – In this state the code is waiting for an IP Address, so the next token will be saved in the event that the client-hostname of this lease matches the one I am looking for. Move to state 2.
- State 2 – The next character should be an open curly bracket. If that’s what `GetToken` returns then the code moves to state 3, otherwise it returns to state 0 and resumes waiting for a lease.
- State 3 – At this point the code is expecting to find statements or a close curly bracket. The statement the code is looking for is "client-hostname "office";". If `GetToken` returns "client-hostname" it moves to state 5, otherwise it moves to state 4.
- State 4 – This statement didn’t start with "client-hostname" so the code keeps reading tokens until it finds one delimited by a semicolon (which indicates the end of a statement.) If it finds one it moves back to state 3; otherwise it remains at state 4.
- State 5 – The code found "client-hostname" so it needs to compare the next token with the client it is looking for. If it matches it saves it. Regardless of whether it matches or not, the code moves to state 6.
- State 6 – The code has found everything of interest in this lease, so it waits untill `GetToken` returns a close curly bracket, indicating the end of the lease declaration. If found, the code returns to state 0, otherwise it remains at state 6.

Now that the states have been determined, all that’s left is some fairly straightforward coding. Here’s the loop (the state variable has a starting value of 0):

Listing 5.

```
LOOP
  token &= GetToken(fin, delimiter)
  IF Token &= NULL
    BREAK
  END
  CASE state
  OF 0
    ! Start and wait for a lease declaration
    IF UPPER(token) = 'LEASE'
      state = 1
    END
    DISPOSE(token)
  OF 1
    ! Found a lease, the next token should be an IP address
    ! If I were being rigorous, I'd probably want to look at this
    ! token and ensure that it at least resembles the format of an
    ! IP address
    IF NOT saveIpAddress &= NULL
      DISPOSE(saveIpAddress)
      saveIpAddress &= NULL
    END
    saveIpAddress &= token
    state = 2
  OF 2
    ! I've gotten what should be an IP address,
    ! the next token should be an open curly brace
    IF token = '{{'
      state = 3
    ELSE
      ! Didn't find an open curly brace, go back to looking
      ! for the start of a lease declaration
      state = 0
    END
    DISPOSE(token)
  OF 3
    ! After finding an open curly brace,
    ! or after a later statement ends with a ';'
    ! wait for "client-hostname'
    IF token = '}'
      state = 0
    ELSIF UPPER(token) = 'CLIENT-HOSTNAME'
      state = 5
    ELSE
      state = 4
```

```
        END
        DISPOSE(token)
      OF 4
        IF token = '}'
          state = 0
        ELSIF delimiter = VAL(';')
          ! Statement has ended, go back to looking for "client-hostname"
          state = 3
        END
        DISPOSE(token)
      OF 5
        ! This is the token found after "client-hostname"
        IF token = '}'
          state = 0
        ELSIF UPPER(token) = UPPER(pClientName)
          ! This is the client I am looking for, save the IP address
          ! But there could be more instances, so keep going and save
          ! the latest one.
          IpAddress &= saveIpAddress
          saveIpAddress &= NULL
          state = 6
        ELSE
          state = 6
        END
        DISPOSE(token)
      OF 6
        ! I've found the client-hostname I am looking for,
        ! so wait for the end of the lease declaration
        IF token = '}'
          state = 0
        END
        DISPOSE(token)
      END
    END
```

FSM3.zip contains a complete implementation of the finite state machine. As you can see the FSM code itself is quite simple and easy to write; the work is in determining the potential states and the data which triggers a change from one state to another.

While this example is fairly simple, I hope it illustrates how you might use finite state machine methodology for solving a variety of coding challenges. I find it to be another way to approach a problem when I ‘m thinking "When I look at this *I* know how to do it, but how do I tell the computer how to do it?"

Tips

- When your state machine grows to more than 5 or 6 states, give the states `EQUATE`d names, like `WAIT_FOR_LEASE` or `WAIT_FOR_END_OF_LEASE`. I find it improves the readability immensely.
- Make sure each state has an exit condition even when there is unexpected or corrupted input, otherwise you can "hang" in a state.

Download the source

John Christ is an Electrical Engineer turned C programmer who was dragged kicking and screaming into the Clarion world around 1995. He wrote much of the near universally reviled pre-C6 Clarion Version Control System. In addition to Clarion, he continues to develop in C/C++, VB.NET and C# on the Windows, Palm and Pocket PC platforms.

Reader Comments

Add a comment

Clarion Magazine

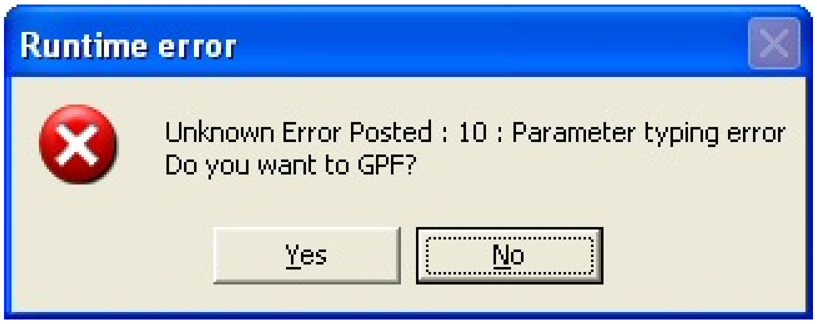
Adding Arrays To Generic Queues With HOWMANY

by Alan Telford

Published 2006-11-17

Several years ago I wrote an article called Debugging Queues with Excel in which I described how to export the contents of a queue to Excel for easy viewing. I was extremely pleased by the great feedback I got from readers.

BUT (there's always a but …) I've always been aggravated by one limitation. My code exported the queue contents as a CSV file, and my generic `ExportQtoCsv` procedure would not work if the queue contained any arrays. When I forgot about this limitation I would see the error in Figure 1.



Unknown Error Posted : 10 : Parameter typing error
Do you want to GPF?

Figure 1. The parameter typing error

I avoided arrays in queues as much as possible, but when I did need arrays and wanted to debug my queue I had to create a new queue which mimicked the original but flattened the array (and I had to write the code to copy from `ArrayQ` to `FlattenedQ`):

```
ArrayQ to FlattenedQ):
ArrayQ      queue
field1        long
field2        long,dim(3)
            end
```

```
FlattenedQ  queue
field1        long
field2_1      long
field2_2      long
field2_3      long
            end
```

Enough is enough!

This was too much work. How could I call myself a lazy programmer if I didn't stop this wasted effort? Fortunately Clarion 6 had arrived and with it a new language function - `HOWMANY(` *`label ,element`*`)`.

- `HOWMANY(ArrayQ,1)` returns 1 as `ArrayQ.field1` is not an array.
- `HOWMANY(ArrayQ,2)` returns 3 as `ArrayQ.field2` is an array with three dimensions.

Here is the original code, which takes a single record from `myQueue` and writes the contents into the string `Line`:

```
Line = ''
Ndx = 0
loop
  Ndx += 1
  AnyVar &= WHAT( myQueue, Ndx)
  if AnyVar &= Null then break.
  Line = Line & choose(~Line,'', ',') & AnyVar
End
```

I need two new variables:

```
HowmanyCnt  long
HowmanyNdx  long
```

I also need an `IF` statement with an embedded loop (eight more lines of code):

```
Line = ''
Ndx = 0
loop
  Ndx += 1
  AnyVar &= WHAT( myQueue, Ndx)
  if AnyVar &= Null then break.
  HowmanyCnt = howmany(myQueue, ndx)
```

```
    if HowmanyCnt = 1
      Line = Line & choose(~Line,'', ',') & AnyVar
    else
      loop HowmanyNdx = 1 to HowmanyCnt
        AnyVar &= WHAT( myQueue, Ndx, HowmanyNdx)
        Line = Line & choose(~Line,'', ',') & AnyVar
      end
    end
```

I check each field in the queue for the number of dimensions. *One* dimension means a normal field and is already handled by the existing code. *More than one* dimension indicates an array field. This has an inner loop to traverse the array using WHAT to get each field within the array.

Extending GroupToINI

You know what it's like. As soon as you learn to use a new feature, you immediately see other uses for it. In September Jeff Slarve wrote an article showing how to store a group in an INI file. His function didn't support arrays either but I had to satisfy myself that I could remove that limitation. The new lines are in bold:

```
GroupToIni   Procedure(*Group pG,String pSection,|
                     String pINIFile,Byte Direction=1)
Ndx LONG
A   ANY
HowmanyCnt  long
HowmanyNdx  long
  Code
  Ndx = 0
  Loop
    Ndx += 1
    A &= WHAT(pG,Ndx)
    If A &= NULL then break.
    HowmanyCnt = howmany(pG,Ndx)
    if HowmanyCnt = 1
      Case Direction
      of 1
        PutINI(pSection,WHO(pG,Ndx),A,pINIFile)
      of 2
        A = GetINI(pSection,WHO(pG,Ndx),,pINIFile)
      end
    else
      loop HowmanyNdx = 1 to HowmanyCnt
        A &= WHAT(pG,Ndx,HowmanyNdx)
```

```
              Case Direction
              of 1
                PutINI(pSection,WHO(pG,Ndx)&'_'&HowmanyNdx,A,pINIFile)
              of 2
                A = GetINI(pSection,WHO(pG,Ndx)&'_'&HowmanyNdx,,pINIFile)
              end
            end
          end
        end
        A &=NULL
```

The only significant difference is that this code appends the dimension count `WHO(pG,Ndx)&'_'&HowmanyNdx` onto the field name.

Summary

Once again I can happily call myself a lazy programmer. My frustration with lack of array support eventually drove me to learn how to use the clarion function `HOWMANY`.

Hmmm … but my new `ExportQtoCsv` and `GroupToINI` still doesn't support embedded groups, and Clarion 6 does have the new `ISGROUP` function. Maybe there is another lazy programmer out there just waiting for the opportunity?

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

Download the source

Alan Telford has been programming in Clarion since 1994. He is the Chief Software Developer at Maxtel Software Ltd, a New Zealand software company specializing in writing back office computer solutions for McDonald's Family Restaurants and other similar markets.

Reader Comments

Add a comment

Clarion Magazine

How To Use The Debugger (video)

by Richard Rose

Richard Rose's Camtasia presentation on how to use the debugger is now available to ClarionMag subscribers by special arrangement with the UK Clarion User's Group. This is a 26 minute, 33 meg AVI file. If you have trouble downloading please delete any partial downloads and do a Save As.

Please post any comments below.

Download the video

Reader Comments

Add a comment

- » Is this a visual presentation because all I get is voice...
- » Sorted out
- » If you need codecs for your Windows Media Player download...
- » Only getting voice, no video ... and W-MP Properties...
- » Download the codecs (see above comment) and see if that...

Clarion Magazine

Threads: When START Starts

by Steven Parker

Published 2006-11-10

Dave Harms recently observed, "One of the most-anticipated features of Clarion 6 is also one of the least-used." That feature is, in the final analysis, preemptive threading (his article shows how to do preemptive threading inside OOP). There have been a number of articles written about the gotcha's and how-to-deal-with's of preemptive threading. I have, myself, made some contribution to this. Dave's article is one of the few "here is a cool thing you can do with preemptive threads."

Now I want to show you another cool thing you can do with the C6 threading model: control when a `START`ed thread actually starts.

"But Herr Doktor," you may be thinking "when I `START` a procedure, obviously it starts." You would not be the first person I've spoken with recently to think this and, like them, you'd be wrong. And I can prove it.

`Start(NextProc)` does not, in fact, immediately launch `NextProc`. Not only doesn't `NextProc` not launch immediately, the documentation quite clearly says so.

The documentation on the `START` statement includes this:

> Code execution in the launching thread immediately continues with the next statement following the START and continues until an ACCEPT statement executes. Once the launching thread executes ACCEPT, the launched procedure begins executing its code in its new thread, retaining control until it executes an ACCEPT.

Read that carefully. The statement immediately after `Start(NextProc)`, for example, executes immediately, not the first statement in `NextProc`. `Accept`, it seems, adjudicates threading within the RTL.

Let's construct an application to test this.

First, I'll place an item on the main menu, Standard START(). This calls a procedure labeled `StdStart`. Actually, the main menu `START`s `StdStart`.

`StdStart` is a Source template procedure. The entirety of its code is:

```
Start(NextProc)
Stop('I just started NextProc')
```

NextProc is also a Source procedure. Its code is:

```
Message('Hi! This is NextProc')
Return
```

If `Start(NextProc)`is immediately launched both `StdStart`'s `Stop` window and `NextProc`'s `Message` window should be visible simultaneously.

They are not.

You may prove this by running the demo app downloadable at the end of this article. (Written in 9054, the EXE is compiled locally so you do not need any runtimes to see the effects.) If you click Standard START() on the main menu, you will see `StdStart`'s `Stop` window. You will not see `NextProc`'s `Message` window. You will not see it even if you move the `Stop` window out of the way.

You will see the `Message` window when you click "Ignore" on the `STOP` window.

Just as I claimed, `Start()` does not immediately launch the named procedure. The procedure does not launch until `Accept` cycles. In this case, of course, `STOP` suspends the `Accept` loop so `NextProc`, precisely as documented, does not start.

In many, if not most cases, this isn't important. But there are times when I want the called procedure to start right [expletive deleted] *now*! How can I do that?

Resume

When SoftVelocity changed the threading model and rewrote the support code, they introduced a new, thread-related, statement: `Resume`. The Language Reference states:

> The RESUME procedure restores a thread that has been suspended with the SUSPEND statement. If the threadno parameter is a number of a thread that was previously suspended by the call to SUSPEND, its suspending count is decremented. If the suspending count becomes equal to zero(0), execution of the thread continues from the point where it has been suspended. Therefore, the number of calls to RESUME must be equal to the number of calls to SUSPEND for the thread execution to resume.
>
> RESUME can also be used to activate a new thread immediately. Normally, a procedure does not allocate memory for thread variables until the ACCEPT event handler is executed. RESUME can be used to activate a new thread directly upon procedure entry.

Read the second paragraph again.

Suppose I copy `StdStart` to `ResumeStart` and change its code to read:

```
Resume(Start(NextProc))
stop('I just started NextProc')
```

If I understand the documentation correctly, `NextProc` will launch immediately. If `NextProc` launches immediately, then I should see both windows.

This is just what happens (see Figure 1).

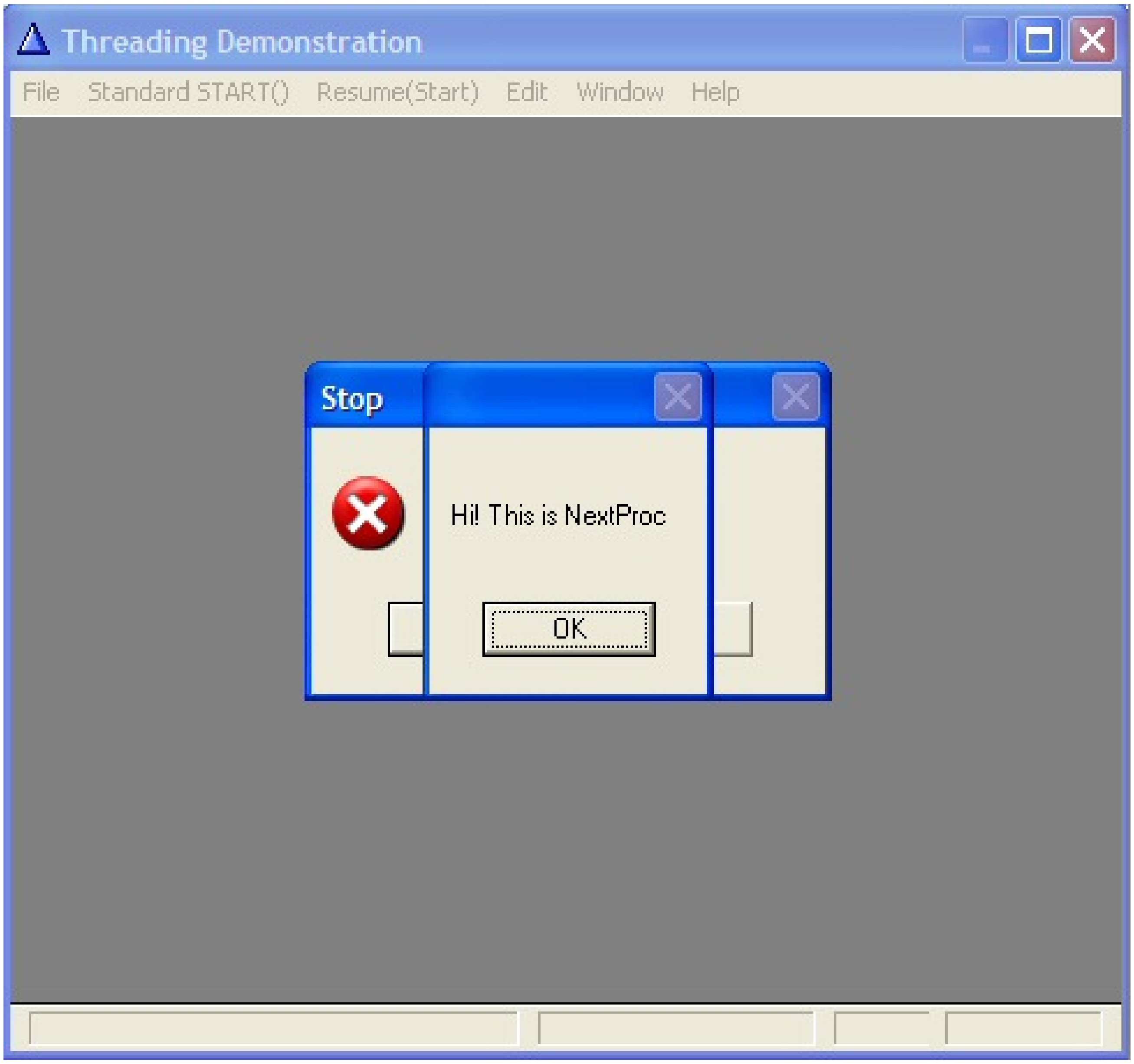


Figure 1. Resume(Start... and both procedures are running

In the demo app, select "Resume(Start)" from the main menu to prove it to yourself.

Summary

In the vast majority of cases, `Accept` cycles frequently enough that the slight delay in launching a new thread is not only unnoticeable, it is irrelevant. However, there are times when "immediately" has to mean "right now, not in a fraction of a second." In fact, it could be more than a fraction of a second. I have a case where I want to start a thread to print part of a report while the remaining code in an embed peforms several hundred computations. In my case, it takes up to four seconds for the remaining code to execute and, therefore, up to four seconds before `Accept` regains control.

Adapting a remark made years ago, in another context: you don't always need to use `Resume` to launch a thread immediately. But, when you do, there's no substitute.

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

- » Wow, I knew you were on the inside track, Steve, but...
- » Thanks, good info. Definitely something to keep in...

Clarion Magazine

Signing Your Applications, Part 2

by Jane Fleming

Published 2006-11-05

In an Internet-connected world, security is becoming increasingly important. How do your users know the application on their computer is the same application you created? The answer is to give each of your EXEs and DLLs an encrypted signature, in a process called *signing*.

In Part One I explained the concepts of encryption and signing; now it's time to put all the pieces together.

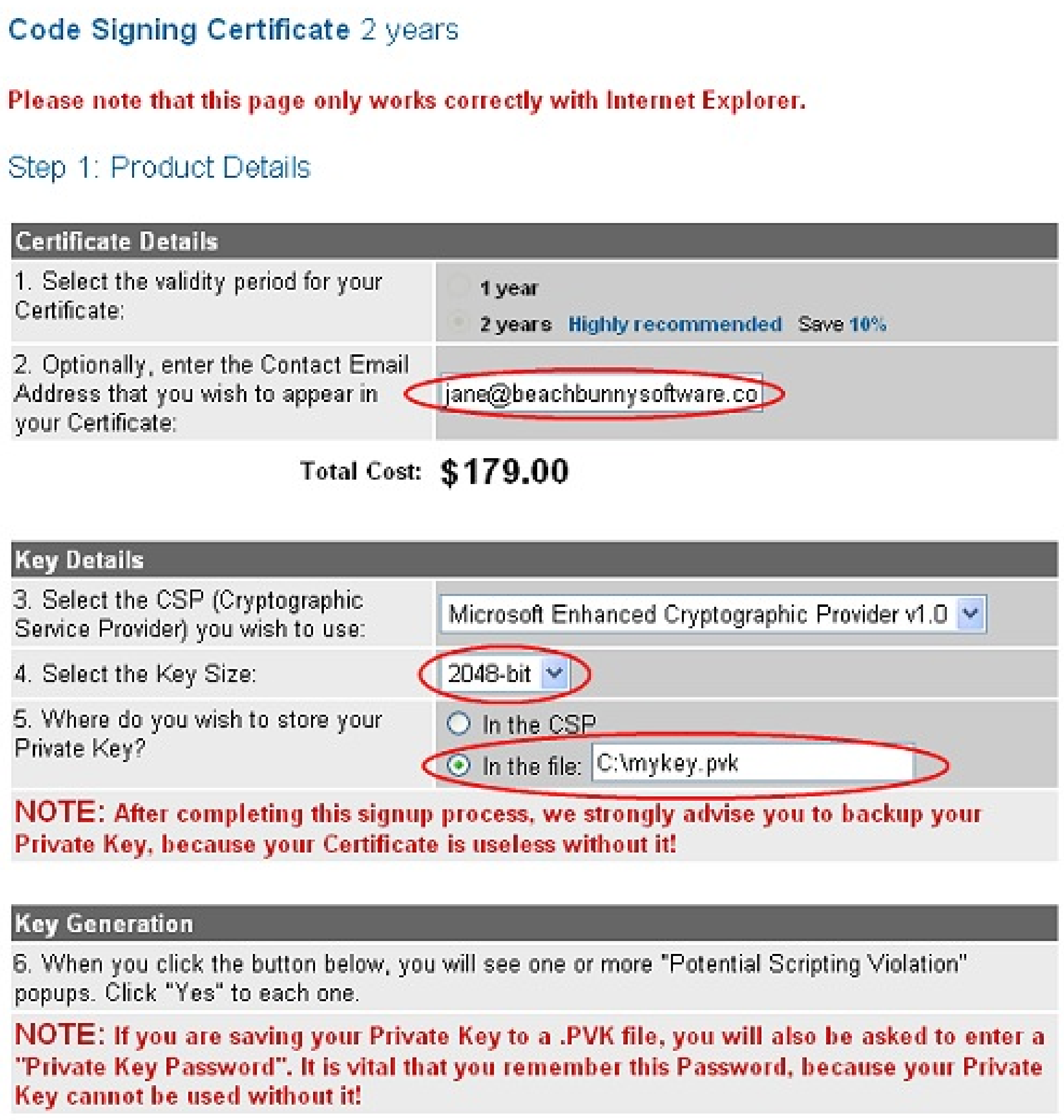
Your digital certificate

As explained in the previous section, it's important to have a digital certificate that is 'signed' by an authority that all your potential clients are likely to recognize.

The best known certification authority is probably Verisign. A code-signing certificate from them will cost you $895 for a two-year validity period. Ouch!

Comodo's certificates are substantially less expensive - $99 for one year or $179 for two years. Don't get lost navigating their website, though. Use this link.

When you purchase your certificate, you'll need to make a couple of decisions. I opted to include an e-mail address, specified a 2048-bit key, and chose to store my private key in a file (Figure 9).



Code Signing Certificate 2 years

Please note that this page only works correctly with Internet Explorer.

Step 1: Product Details

Certificate Details	
1. Select the validity period for your Certificate:	1 year [✓ 2 years] Highly recommended Save 10%
2. Optionally, enter the Contact Email Address that you wish to appear in your Certificate:	jane@beachbunnysoftware.co

Total Cost: $179.00

Key Details	
3. Select the CSP (Cryptographic Service Provider) you wish to use:	Microsoft Enhanced Cryptographic Provider v1.0
4. Select the Key Size:	2048-bit
5. Where do you wish to store your Private Key?	In the CSP [✓ In the file: C:\mykey.pvk]

NOTE: After completing this signup process, we strongly advise you to backup your Private Key, because your Certificate is useless without it!

Key Generation

6. When you click the button below, you will see one or more "Potential Scripting Violation" popups. Click "Yes" to each one.

NOTE: If you are saving your Private Key to a .PVK file, you will also be asked to enter a "Private Key Password". It is vital that you remember this Password, because your Private Key cannot be used without it!

Figure 9. Certificate options (view full size image)

As the screen in Figure 9 illustrates, you'll next need to create a password.

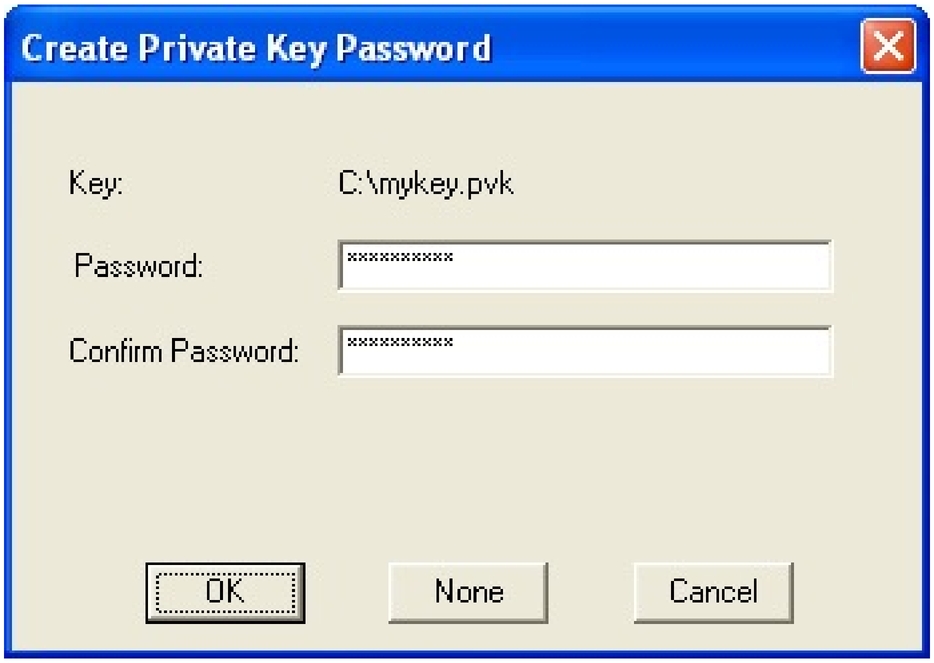


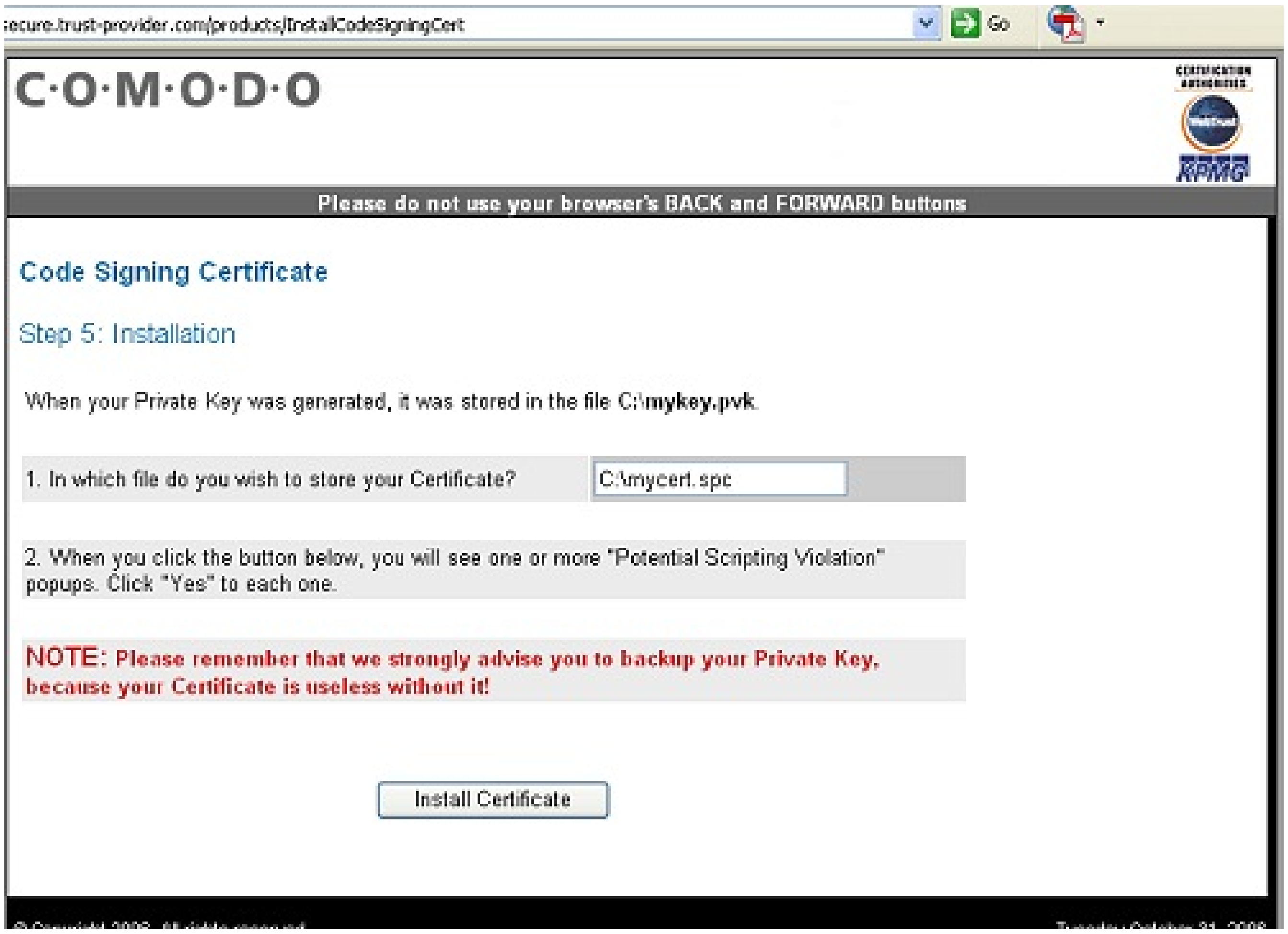
Figure 10. Private key password

Then you'll supply your company and contact information, and pay for the certificate.

For SSL certificates, it is important to spell out (not abbreviate) your state or province. I'm not sure whether that's true for code-signing certificates, but I spelled my state name anyway.

I faxed a copy of my city-issued business license, and had my confirmation e-mail the same day.

Using the information contained in your confirmation e-mail, you'll pick up your certificate. Again, I opted to store mine in a file (Figure 11).



secure.trust-provider.com/products/InstallCodeSigningCert

C·O·M·O·D·O

Please do not use your browser's BACK and FORWARD buttons

Code Signing Certificate

Step 5: Installation

When your Private Key was generated, it was stored in the file C:\mykey.pvk.

1. In which file do you wish to store your Certificate? C:\mycert.spc

2. When you click the button below, you will see one or more "Potential Scripting Violation" popups. Click "Yes" to each one.

NOTE: Please remember that we strongly advise you to backup your Private Key, because your Certificate is useless without it!

Install Certificate

Figure 11. Picking up your certificate (view full size image)

Now might be a good time to copy both the certificate and private key files onto a floppy or CD and to store those along with your password in a safe deposit box or other secure location.

The Signing Software

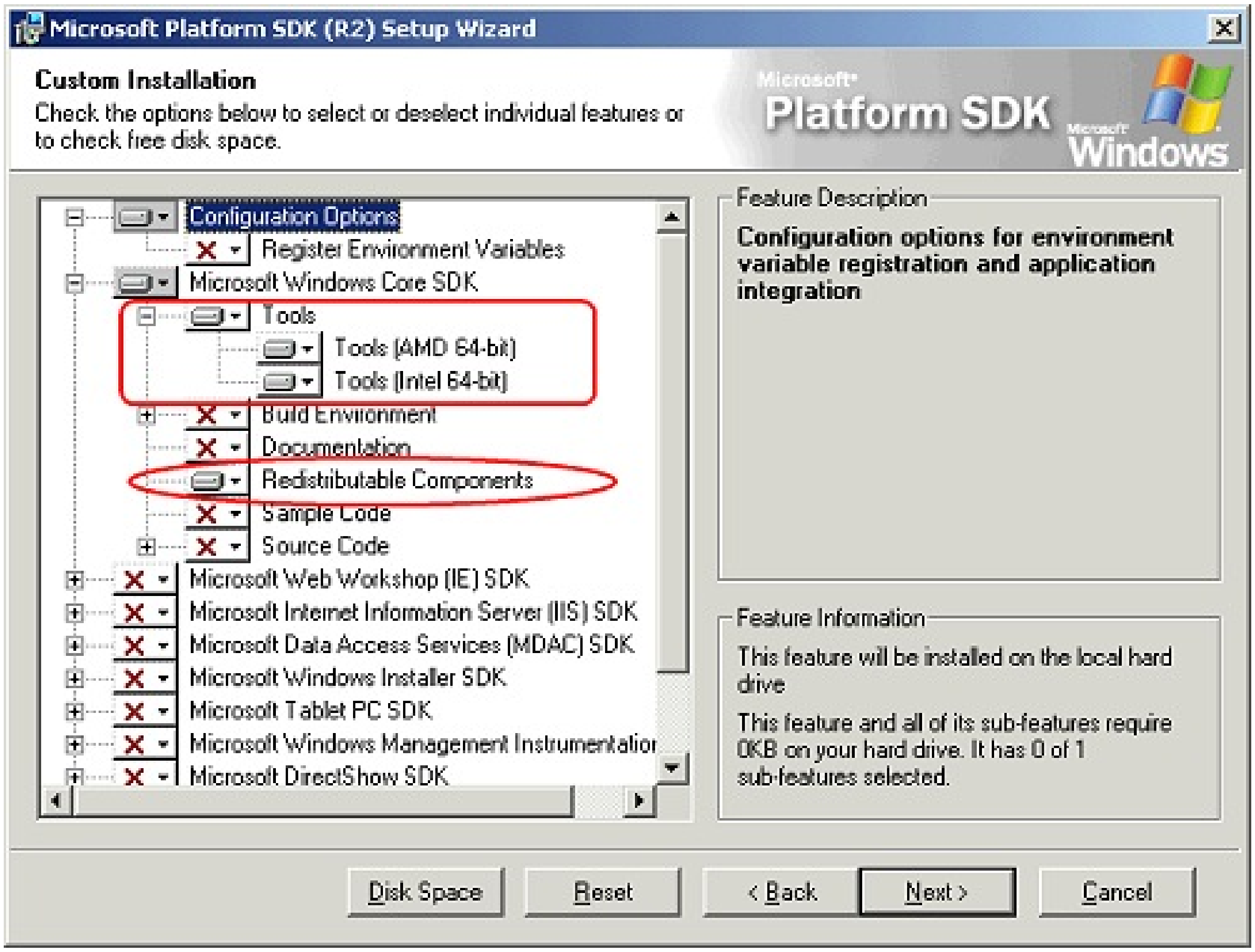
To actually sign your EXEs and DLLs you'll use a couple of small utilities from Microsoft, but you'll have to install a current version of the Platform SDK to get them. Microsoft's website has a number of references to `signtool.exe`, but many of the links are outdated. I suggest going to microsoft.com and in the search window typing:

```
    server 2003 r2 SDK web install
```

On the search results page, in addition to the web install you'll see links by which you can download the SDK in seventeen .cab files of 25 megabytes apiece or as an image to burn onto a CD. Microsoft also has a CD available for the bandwidth-challenged.

You’re only going to need a minimum of components, but the install still required about 240 MB on my hard drive. I used the web install link (which is why I specified web install in the search string), and started the installation by clicking the psdk-x86.exe button.

Once the installer begins, installing `Tools` and `Redistributable Components` (Figure 12) will suffice.



Microsoft Platform SDK (R2) Setup Wizard

Custom Installation

Check the options below to select or deselect individual features or to check free disk space.

- Configuration Options
  - Register Environment Variables
- Microsoft Windows Core SDK
  - Tools
    - Tools (AMD 64-bit)
    - Tools (Intel 64-bit)
  - Build Environment
  - Documentation
  - Redistributable Components
  - Sample Code
  - Source Code
- Microsoft Web Workshop (IE) SDK
- Microsoft Internet Information Server (IIS) SDK
- Microsoft Data Access Services (MDAC) SDK
- Microsoft Windows Installer SDK
- Microsoft Tablet PC SDK
- Microsoft Windows Management Instrumentation
- Microsoft DirectShow SDK

Feature Description

Configuration options for environment variable registration and application integration

Feature Information

This feature will be installed on the local hard drive

This feature and all of its sub-features require 0KB on your hard drive. It has 0 of 1 sub-features selected.

Figure 12. Installing portions of the Platform SDK (view full size image)

You’ll be able to delete most of what the install program puts on your disk after the installation is finished.

Signing your code using the wizard

You can sign your code using command line tools, or using Microsoft's wizard. The wizard provides the easiest interaction, but you'll quickly find it annoyingly time-consuming.   Later, I'll show you how to automate the process using a batch file.

The signing program is called `signtool.exe`; you can find it in the `Bin` folder under the folder into which you installed the SDK.

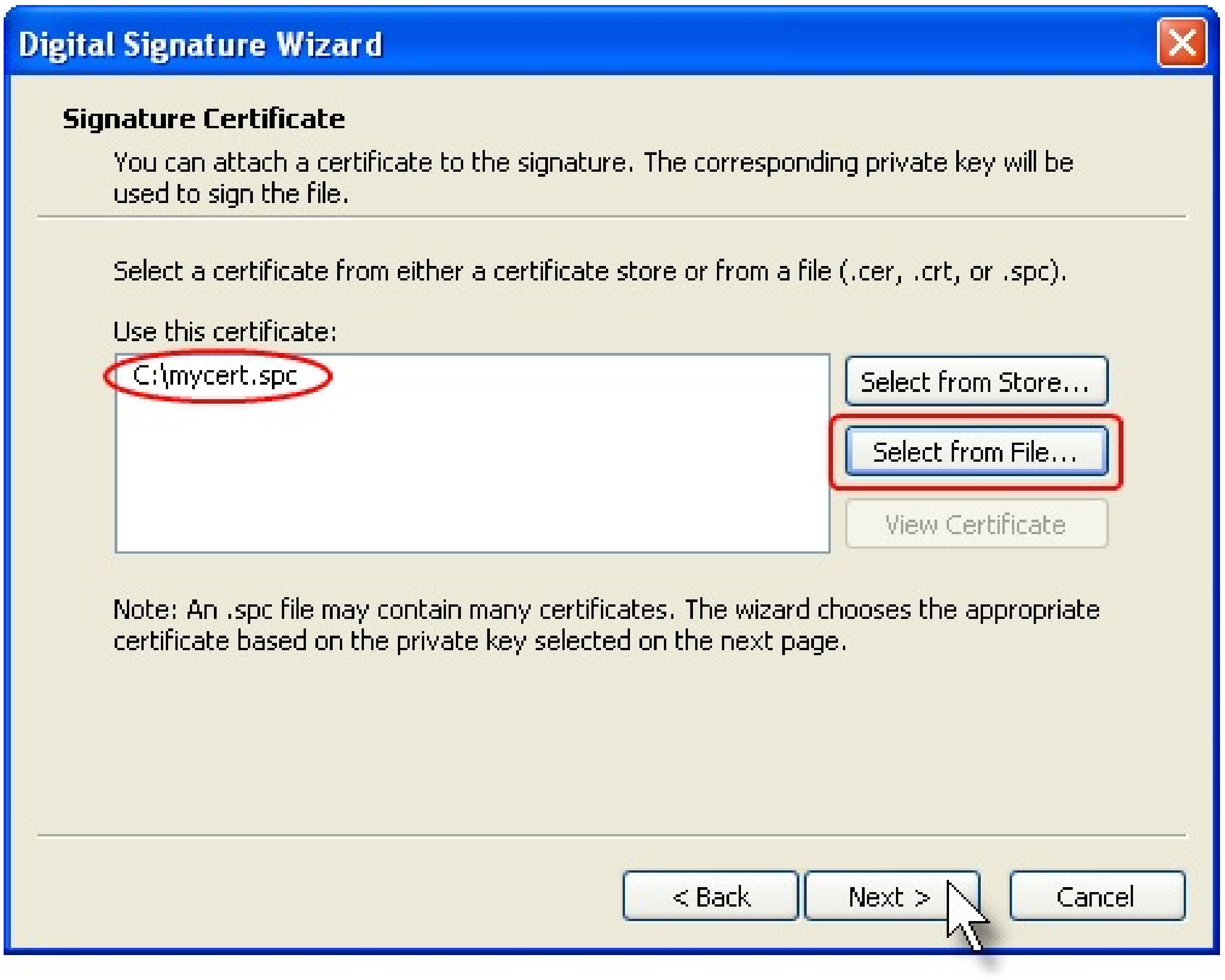
You will need to invoke this tool from a command prompt, or else make several shortcuts using the appropriate command-line switches.

From the command line, navigate to the `Bin` folder and then type

```
signtool signwizard
```

Click through the wizard's welcome screen, then browse and select the program or DLL that you want to sign. On the next screen, click the `Custom` button and then `Next`.

Assuming you saved your certificate in a file, click the `Select from File`... button, then navigate to your certificate file. (Figure 13).


Digital Signature Wizard

Signature Certificate

You can attach a certificate to the signature. The corresponding private key will be used to sign the file.

Select a certificate from either a certificate store or from a file (.cer, .crt, or .spc).

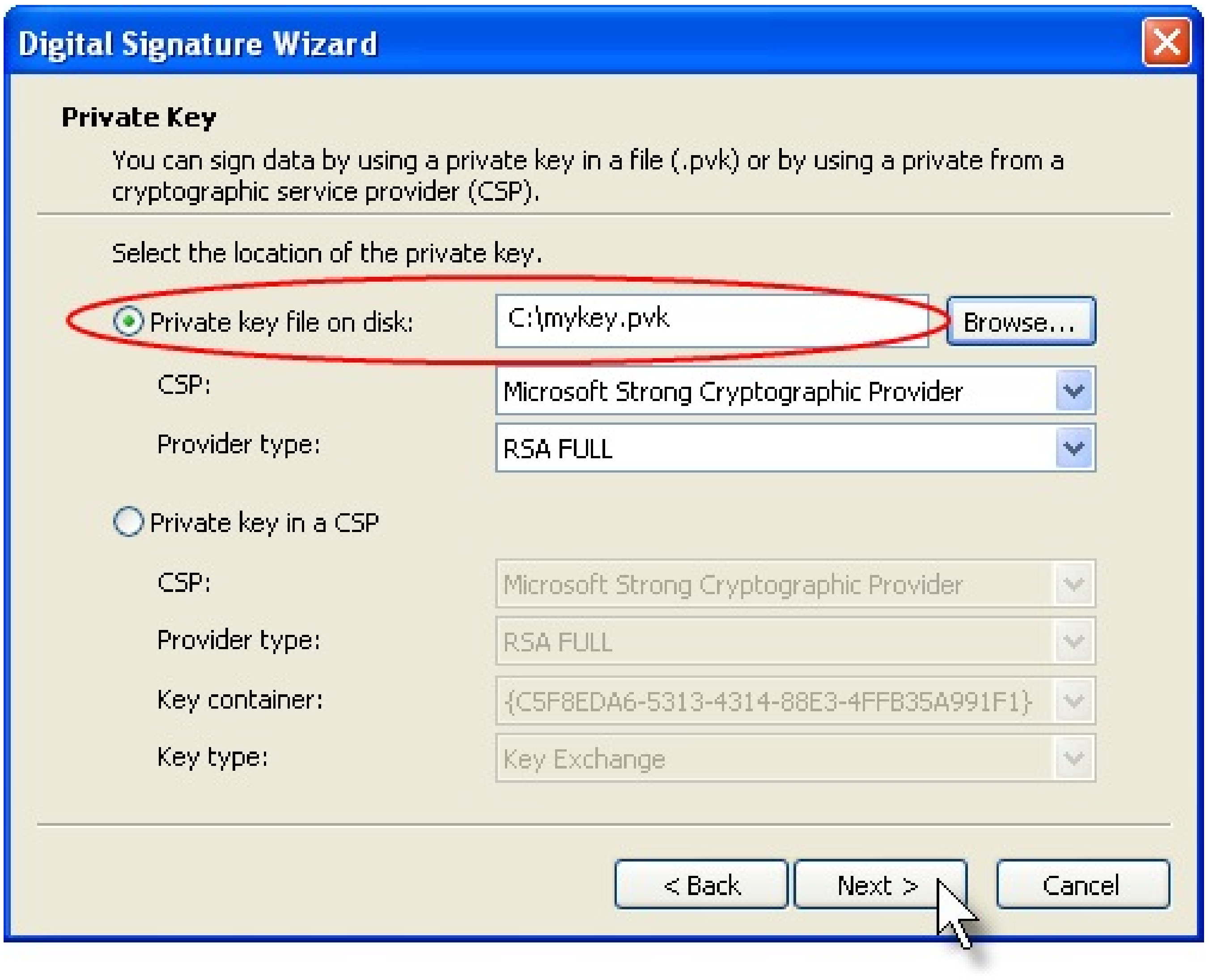
Use this certificate:

C:\mycert.spc

Note: An .spc file may contain many certificates. The wizard chooses the appropriate certificate based on the private key selected on the next page.

Figure 13. Selecting the certificate

On the next screen (Figure 14), select your private key file.



Digital Signature Wizard

Private Key

You can sign data by using a private key in a file (.pvk) or by using a private from a cryptographic service provider (CSP).

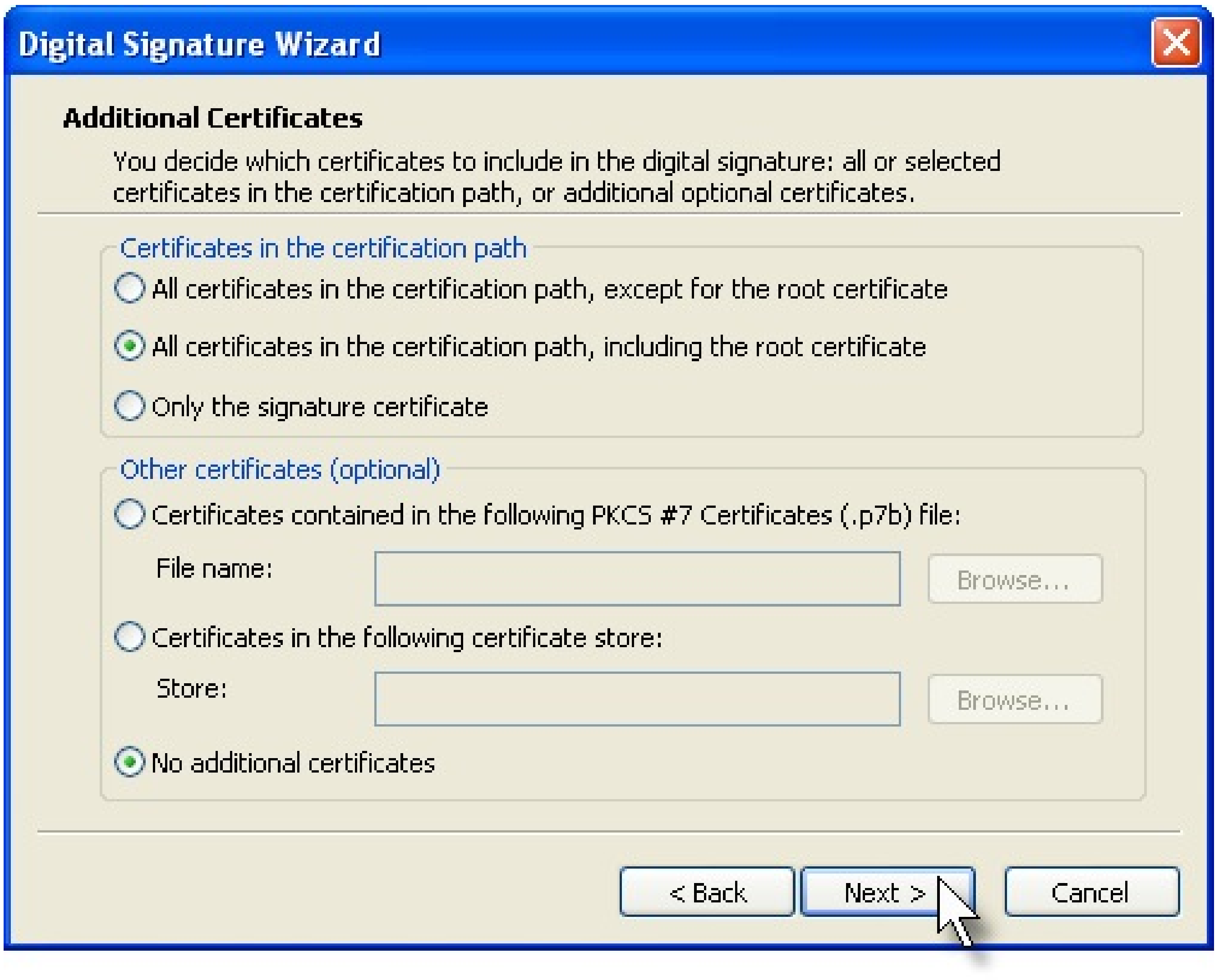
Select the location of the private key.

- (●) Private key file on disk: C:\mykey.pvk
  - CSP: Microsoft Strong Cryptographic Provider
  - Provider type: RSA FULL
- ( ) Private key in a CSP
  - CSP: Microsoft Strong Cryptographic Provider
  - Provider type: RSA FULL
  - Key container: {C5F8EDA6-5313-4314-88E3-4FFB35A991F1}
  - Key type: Key Exchange

Figure 14. Selecting the private key

When you click `Next`, you'll need to enter the password you specified when you created your private key. After doing so, you'll be asked for a hash algorithm. Select the default (`sha1`) and click `Next`.

Accept the defaults on the next screen (Figure 15) and click `Next`.


Digital Signature Wizard

Additional Certificates

You decide which certificates to include in the digital signature: all or selected certificates in the certification path, or additional optional certificates.

Certificates in the certification path

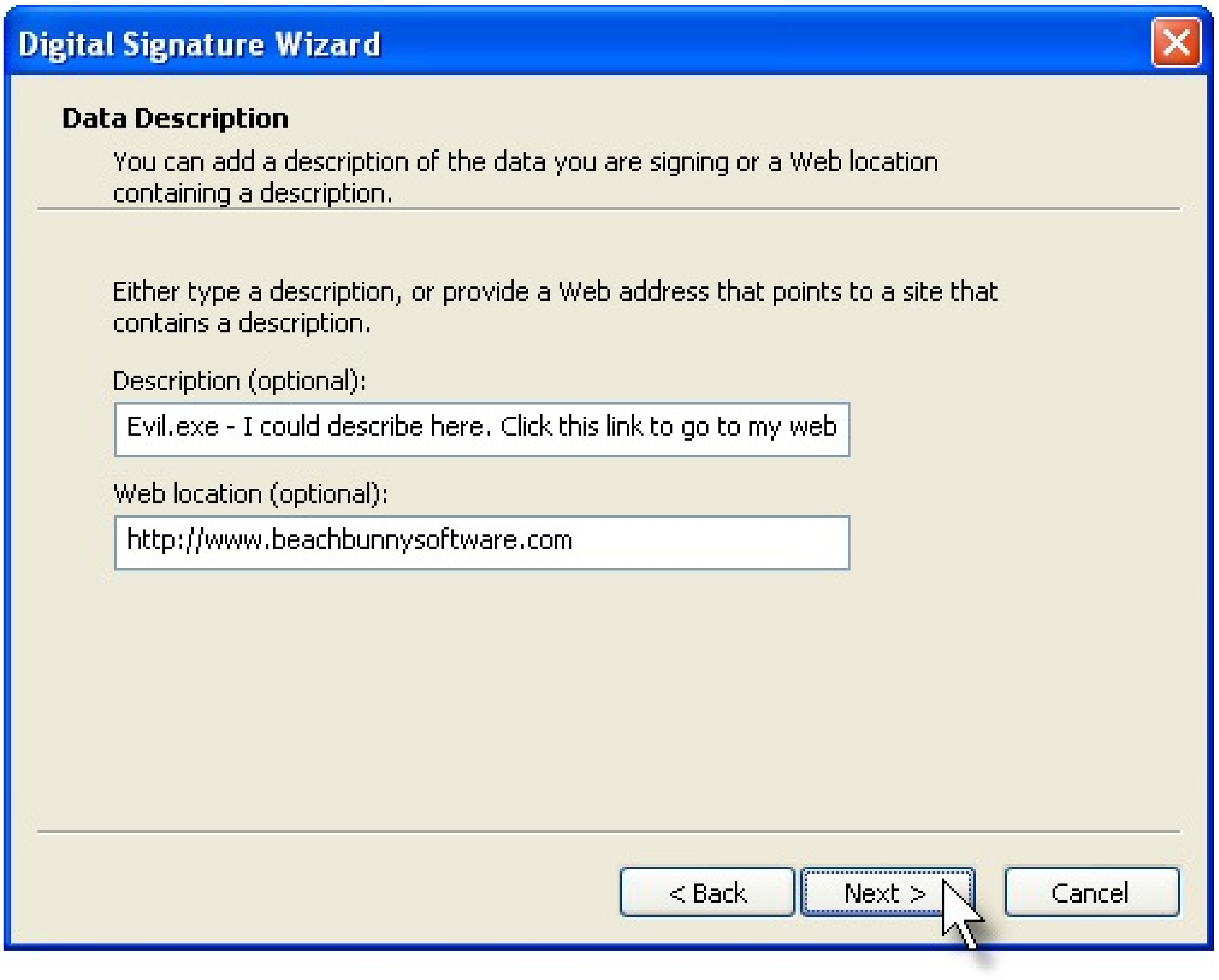
- ○ All certificates in the certification path, except for the root certificate
- [✓ All certificates in the certification path, including the root certificate]
- ○ Only the signature certificate

Other certificates (optional)

- ○ Certificates contained in the following PKCS #7 Certificates (.p7b) file:
  - File name:
- ○ Certificates in the following certificate store:
  - Store:
- [✓ No additional certificates]

Figure 15. Additional certificates (accept the defaults)

On the next screen, you can optionally add a description of your program and a web link. The information I typed in Figure 16 will produce the screen shown in Figure 6 (in Part One) when a user runs my program.



Data Description

You can add a description of the data you are signing or a Web location containing a description.

Either type a description, or provide a Web address that points to a site that contains a description.

Description (optional):

Evil.exe - I could describe here. Click this link to go to my web

Web location (optional):

http://www.beachbunnysoftware.com

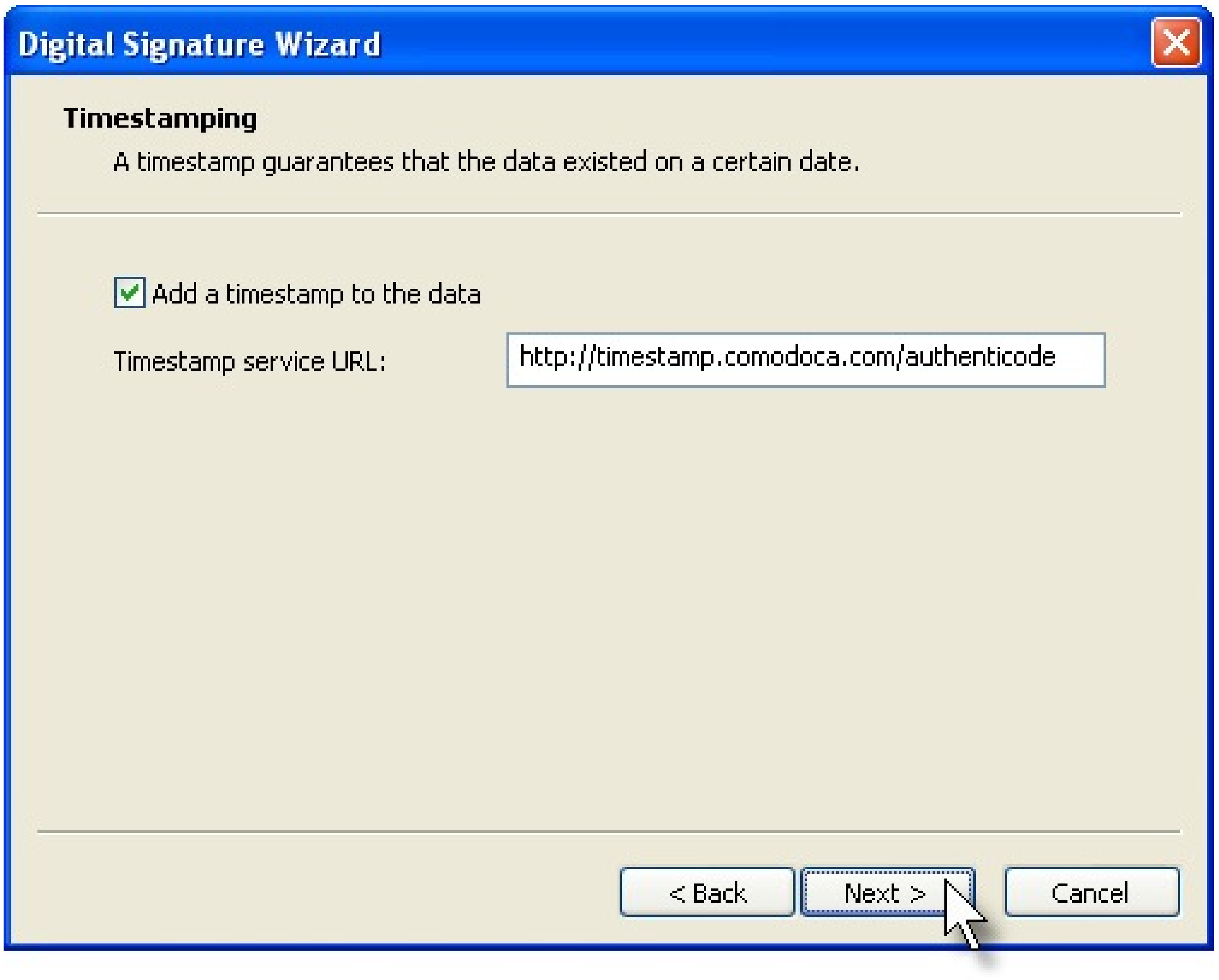
Figure 16. Optional description and web link

The next screen (Figure 17) gives the option of time stamping your signature. (I discussed the purpose of time stamping in Part One.)

Two URLs that you can specify for time stamping are:

- `http://timestamp.comodoca.com/authenticode`
- `http://timestamp.verisign.com/scripts/timstamp.dll` (note that `timstamp.dll` does not contain the letter `e`.)

Yes, Verisign’s time stamp server will countersign a program you’ve signed using a Comodo certificate, if Comodo is your preference.



Timestamping

A timestamp guarantees that the data existed on a certain date.

[✓] Add a timestamp to the data

Timestamp service URL: http://timestamp.comodoca.com/authenticode

Figure 17. Optional time stamp

After you click `Next` from the time stamp screen, you'll have an opportunity to review your choices. Then click `Finish`. You'll again need to enter the password for your private key. Your firewall may notify you that `signtool.exe` is attempting to access the Internet (it's connecting to the time stamp server). Then you should get a message saying that the wizard completed successfully.

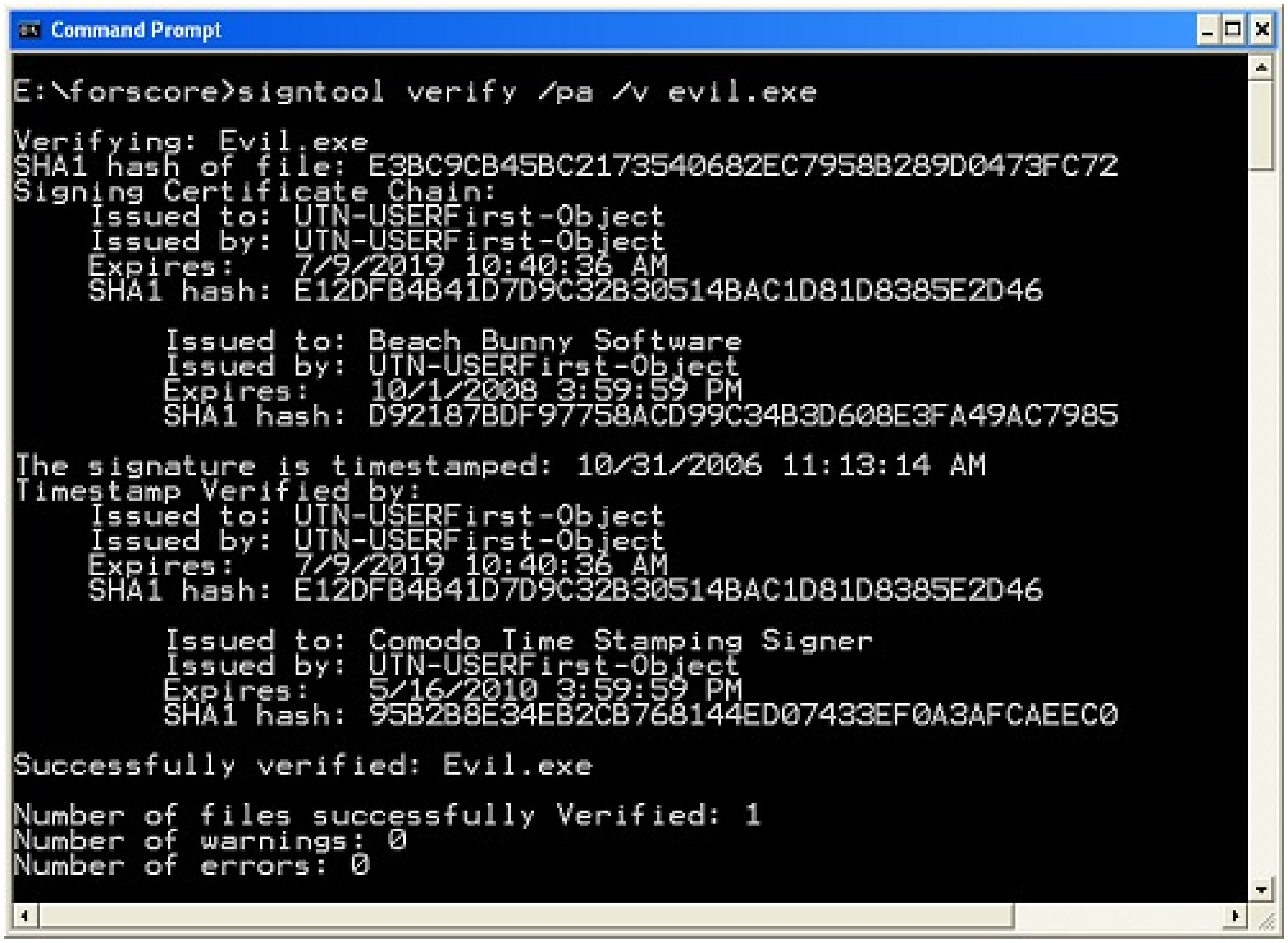
Verifying the Signature

At this point, you have a couple of options for checking your work.

You can use signtool.exe again, with a different command. As the file I signed was called `evil.exe`, the command I would use would be:

```
signtool verify /pa /v e:\forscore\evil.exe
```

Figure 18 shows the result.

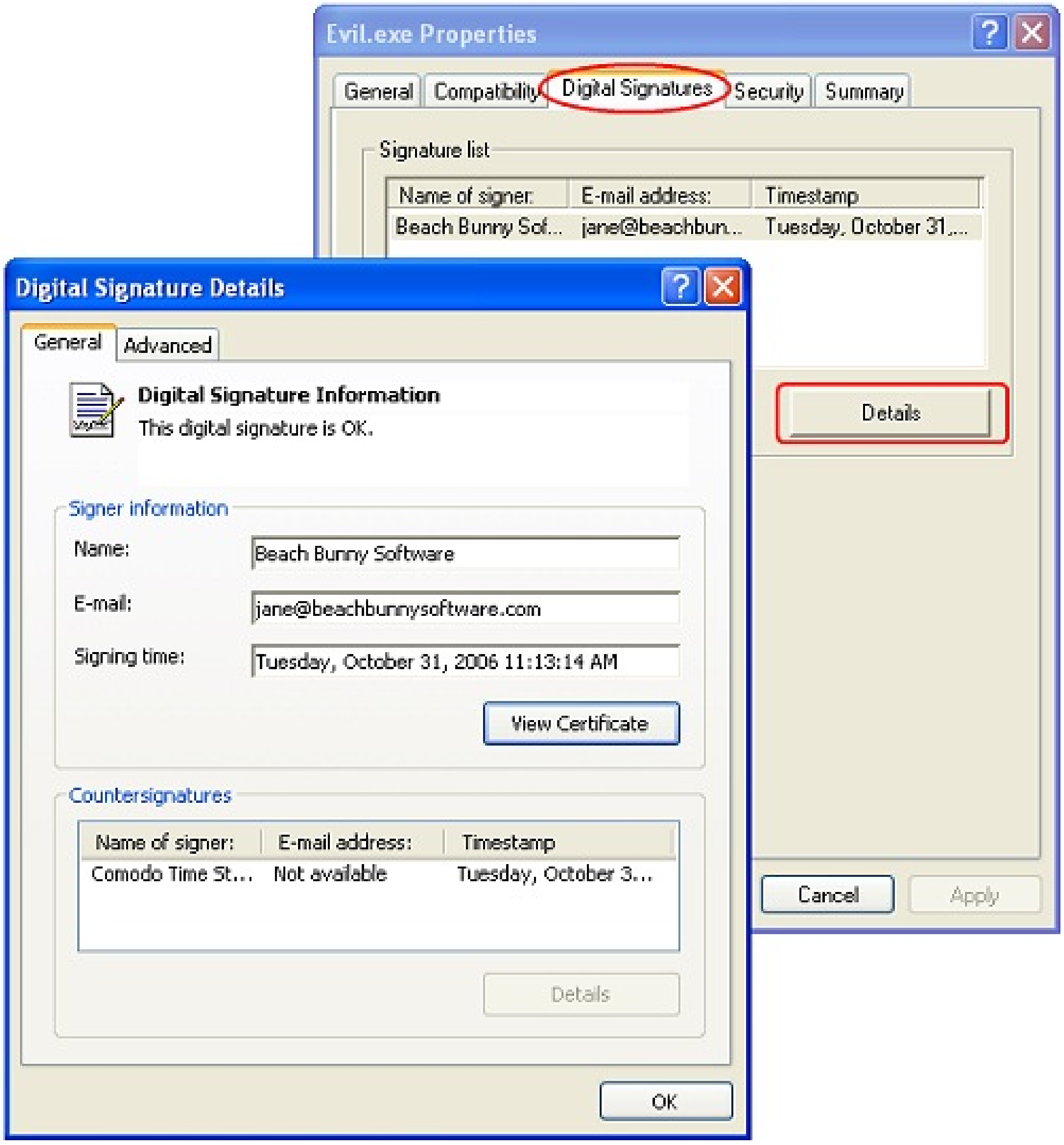


```
E:\forscore>signtool verify /pa /v evil.exe

Verifying: Evil.exe
SHA1 hash of file: E3BC9CB45BC2173540682EC7958B289D0473FC72
Signing Certificate Chain:
    Issued to: UTN-USERFirst-Object
    Issued by: UTN-USERFirst-Object
    Expires:   7/9/2019 10:40:36 AM
    SHA1 hash: E12DFB4B41D7D9C32B30514BAC1D81D8385E2D46

        Issued to: Beach Bunny Software
        Issued by: UTN-USERFirst-Object
        Expires:   10/1/2008 3:59:59 PM
        SHA1 hash: D92187BDF97758ACD99C34B3D608E3FA49AC7985

The signature is timestamped: 10/31/2006 11:13:14 AM
Timestamp Verified by:
    Issued to: UTN-USERFirst-Object
    Issued by: UTN-USERFirst-Object
    Expires:   7/9/2019 10:40:36 AM
    SHA1 hash: E12DFB4B41D7D9C32B30514BAC1D81D8385E2D46

        Issued to: Comodo Time Stamping Signer
        Issued by: UTN-USERFirst-Object
        Expires:   5/16/2010 3:59:59 PM
        SHA1 hash: 95B2B8E34EB2CB768144ED07433EF0A3AFCAEEC0

Successfully verified: Evil.exe

Number of files successfully Verified: 1
Number of warnings: 0
Number of errors: 0
```

Figure 18. Using signtool to verify the signature (view full size image)

You can also use Windows Explorer to check the signature.

Navigate to the file you've signed, right-click it, click `Properties`, and select the `Digital Signatures` tab. You can now click on the signer and then click the `Details` button to view the signing certificate and information (Figure 19).



Name of signer:	E-mail address:	Timestamp
Beach Bunny Sof...	jane@beachbun...	Tuesday, October 31,...

Digital Signature Information
This digital signature is OK.

Signer information

Name:	Beach Bunny Software
E-mail:	jane@beachbunnysoftware.com
Signing time:	Tuesday, October 31, 2006 11:13:14 AM

Countersignatures

Name of signer:	E-mail address:	Timestamp
Comodo Time St...	Not available	Tuesday, October 3...

Figure 19. Verifying the signature in Windows Explorer (view full size image)

Signing your code by batch file

Now that you've accustomed yourself to the trauma of wandering around the command prompt, you're probably ready to automate the signing process.

Your first job is to create a single file that combines your digital certificate and your private key. The tool you'll use is `pvk2pfx.exe`. It's also in the `Bin` folder where you installed the Platform SDK.

To give myself a little legroom, I made a new folder, into which I copied `pvk2pfx.exe`, `mycert.spc`, and `mykey.pvk`. You may want to do likewise.

Assuming the password for my private key is `janepassword`, and that I want a different password (`mynewpass`) for use with my batch file, the command is:

```
pvk2pfx –pvk mykey.pvk –pi janepassword –po
  mynewpass –spc mycert.spc –pfx Jane.pfx
```

Line break added. (Note that there is a space between each switch and its parameter. Although the text wraps in this article, it should all be typed on one line, after which you press `Enter`.).

- `–pi` specifies the input password (the password I typed in when I bought my certificate)
- `-po` specifies the output password (the password that my new .pfx file will use)
- `-pvk` and `–spc` specify my private key and certificate files
- `-pfx` specifies the name of the .pfx file I want to create.

The rest is easy!

Creating the Batch file

Copy your new `PFX` file into your development folder. You may also want to copy `signtool.exe` into that folder, just to make things simpler.

Next, use Notepad to create a batch file called `signstuff.bat`. (As you're probably aware, Notepad may want to name that file signstuff.bat.txt unless you put the file name within quotation marks when you save it.)

My batch file looks like this:

```
@echo off
signtool sign /f Jane.pfx /p mynewpass
/t http://timestamp.comodoca.com/authenticode
```

```
    /d "My Groovy Program"
    /du http://www.beachbunnysoftware.com/idpa/
    /v f*.exe uncommit.exe scr1bx.dll scr1cx.dll scr1dx.dll
```

This file is only two lines long. The second line wraps in this article, but it should be all one line when you create it. Each switch after the word `sign` consists of a forward slash and one or two letters, a space, and then the parameter that switch is specifying. You'll recognize that you're providing pretty much the same information as you did when you used the wizard interface to sign a program.

The first line of the batch file tells the batch processor not to echo commands to the screen.

The second line does the signing:

- It invokes signtool.exe with the `sign` command
- `/f` specifies the signing file (The `PFX` file you created. Mine is `Jane.pfx`)
- `/p` specifies the password for the `PFX` file, the one you specified when you created that file
- `/t` specifies the time stamp option, and is followed by the URL of a time stamp server.
- `/d` is a description of the EXE/DLL being signed ("My Groovy Program") If the description has spaces, be sure to enclose it in quotation marks.
- `/du` is the URL a user can click to get more information about the signed material.
- `/v` asks for verbose output as the signing process runs
- At the end of the batch file's command line is a list of the files to be signed. You can use wild cards and/or specific names. My batch illustration signs all exe files beginning with the letter `f`, a specific executable (`uncommit.exe`), and three specific DLLs.

Conclusion

Signing your software projects is one important step you can take toward increasing user confidence and security. Signing may seem complicated, but once you've assembled your certificate and a few software tools and have learned how to write a simple batch file, signing your project(s) can become a very simple process.

Additional Reading

- Authenticode 101 from Microsoft
- Authenticode 102 from Microsoft
- Windows 2000 and 2003 Certificate Servers
- The SSL handshake
- Comodo’s (somewhat outdated) overview of the signing process
- Wikipedia’s explanation

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

- » I like to make a batch file "sign1xxx.bat for each project...
- » Carl, In real life I also use multiple batch files,...
- » Great articles and very timely, Jane. Thanks. Great bio...

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

Turkey Day, and a Friday bonus...

Direct link

Posted Thursday, November 23, 2006 by Dave Harms

It's American Thanksgiving Day today, and walking through the newsgroups I hear little else but the echo of my footsteps. Seems everyone has gone home for turkey and trimmings! If you're among that number, may your celebration be full and fine.

And in honor of the shopping mayhem that ensues on Black Friday, look in the ClarionMag store tomorrow for a treat or two...

Source code library close to gold release

Direct link

Posted Wednesday, November 22, 2006 by Dave Harms

If you've downloaded the latest release of the source code library you'll need to make one small change to your ClarionMagazineAccessories.ini file. The version line should look like this:

```
    Source Code Library Install=2006.11.21
```

After that it should show up in your desktop. I've also set up an XML feed so the Clarion Desktop server can pick up the current version automatically.

The INI file change will be in the next release, which will probably go up early next week. Right now you need to download the entire install each time there's a new release, but this will change in the new year. I anticipate a Nov 30 release, and then a Dec 31 release, which will be full installs. In the new year there will be two installs, the up-to-2006 version, and then a current 2007 patch, updated monthly. Actually patch is the wrong term, since to keep things simple the 2007 version will just install files overtop of the existing install. At the end of 2007, the main install will be updated with the year's source files, and so on.

I expect the source library beta to wrap up this month, after which prices go up! Remember that for current magazine subscribers the source library is a one time fee, for all others an annual fee. Details.

A finite what machine?

Direct link

Posted Wednesday, November 22, 2006 by Dave Harms

You may not know what a Finite State Machine is, but chances are you've already used on in your code. John Christ has a nice article on the subject and he's been waiting patiently for it to appear; and I hope to have it up by Friday. The article would be up already but I've been busy this week polishing up the Clarion Magazine Source Code Library, which is now packaged in a SetupBuilder install for C3PA and Clarion Desktop compatibility.

The great server migration

Direct link

Posted Wednesday, November 01, 2006 by Dave Harms

The thunder of thousands of servers galloping across the plain, their little server feet pounding the ground into dust...it's server migration time!

A little over three years ago I moved Clarion Magazine to a dedicated server at EV1Servers. Although I've had computers last longer than three years, this summer seemed like a good time to consider an upgrade. And although it's gone fairly smoothly, it hasn't been without problems.

There have been some configuration glitches and I still have a few automated processes to set up. On the whole, the new server seems to be working well.

The new site *should* be identical to the old one, except that I'm using a very slightly newer version of Resin, the application server on which the magazine is built. And it appears that this version of Resin, despite its using exactly the same configuration file as the old server, is applying a different character encoding when it generates the pages. And since this is happening at page generation time, the answer isn't as simple as changing the HTML page's own encoding directive. Mainly this results in the display of some garbled punctuation characters.

I'm on the road right now and my time is somewhat constrained, so it may be a few days before everything's sorted out.