Clarion Magazine

Clarion News

- » First DerivedShell Product Released
- » iQ-XML 1.03
- » ClarionMag/Podcast Coffee Mugs!
- » ClarionMag E-Book RSS Feed
- » New Edit-In-Place E-Book
- » vuAgent Updated
- » DynaLib 4.0.1
- » iQ-XML Updated
- » New Data Ferret Website
- » Solace Flatten Free Template
- » EasyCOM2INC 2.02
- » Easy3DStyle 2.00
- » Free Tech Books #3
- » Free Tech Books #2
- » Free Tech Books #1
- » CapeSoft At South American DevCon
- » Replicate Jumps Forward
- » HotUpDates
- » Secwin Online Server 1.03 Beta Released
- » EasyCOM2INC 2.02
- » XML Parser/Writer Function Library
- » SB5 Developer Edition RC1 Build 1101
- » solid software support schedule
- » KwikSYSTEMS & Comsoft7 Exclusive AppShell (Duke Shells) Source
- » ABCFree Templates and Tools 04/06/2005

- » xPathManager 1.2
- » ClarionMag Free EBook Offer Extended To Friday, April 8
- » ClarionMag Adds Three New E-Books
- » EasyResizeAndSplit 2.08
- » SoftVelocity Training On Demand
- » In-Memory and IP Driver 2.0 Releases Coming Soon
- » EasyCOM2INC 2.01
- » DCT2SQL Templates Updated

[More news]

Podcast

Political mover and shaker Andrew Guidroz spills the beans on how a custom Clarion app helped him wage a successful election campaign, Dave Harms waxes and wanes poetic on Thunderbird, and Dave and Andrew discuss user interface issues and the new Planet Clarion/Clarion Magazine coffee mug. 00:43:46, 15389K

Listen now

[Track lists, more podcasts]

Latest Free Content

New Clarion Magazine Link Images

Clarion Magazine has a new logo, and new linking images. If you have (or would like to have) a link to Clarion Magazine on your web site, you can get updated images here.

PDF: Using Agile Programming Techniques for the Enterprise Information System : A Case Study

Louis Coraggio and Wayne Lundeberg describe how Clarion is used in an extreme/agile programming environment to create an Enterprise Information System for an ISO 9001 manufacturing firm. An overview of the EIS development process, the system design goals, and a chronological narrative of EIS development are presented. Included are additional requirements and recommendations for those considering agile methods.

[More free articles]

Latest Subscriber Content

New Clarion Magazine Link Images (free article)

Clarion Magazine has a new logo, and new linking images. If you have (or would like to have) a link to Clarion Magazine on your web site, you can get updated images here.
Posted Thursday, April 07, 2005

PDF for March 2005

All Clarion Magazine articles for March 2005 in PDF format.
Posted Friday, April 08, 2005

Limit An App To A Single Instance: DDE Strikes Back

In days of old, when programmers were bold and DDE was the ultimate tool, if one wished to limit one's application to a single instance, it was quite easy. But Microsoft has deprecated DDE, and has done its best to move programmers to other solutions. As a result, that old DDE instance-limiting code doesn't always work as expected. Unless you add a modern twist, as Steve Parker shows.
Posted Tuesday, April 12, 2005

PDF: Using Agile Programming Techniques for the Enterprise Information System : A Case Study (free article)

Louis Coraggio and Wayne Lundeberg describe how Clarion is used in an extreme/agile programming environment to create an Enterprise Information System for an ISO 9001 manufacturing firm. An overview of the EIS development process, the system design goals, and a chronological narrative of EIS development are presented. Included are additional requirements and recommendations for those considering agile methods.
Posted Wednesday, April 13, 2005

Version Control with CVS and Clarion 6.x

Recently, Nardus Swanevelder introduced the open source CVS version control system to ClarionMag readers. Now Bernie Grosperrin advances this topic, showing how to use CVS with Clarion 6.x. Part 1 of 2.
Posted Friday, April 15, 2005

Version Control with CVS and Clarion 6.x, Part 2

Recently, Nardus Swanevelder introduced the open source CVS version control system to ClarionMag readers. Now Bernie Grosperrin advances this topic, showing how to use CVS with Clarion 6.x. Part 2 of 2.
Posted Thursday, April 21, 2005

Providing Good Customer Support

They call day and night, email you even on weekends, and actually expect you to help... customers! Arggh! But without them, you're out of business. So what is the best way to manage tech support? In this article Drew Bourrut defines good tech support, and suggests ways to assure that type of support.
Posted Friday, April 22, 2005

Planet Clarion Transcript: Clarion.NET, and Trial Versions

In this prequel to the previous podcast's Bob Z interview, Dave Harms talks with Bob about Clarion.NET, and Andrew and Dave talk about the value of trial editions.
Posted Monday, April 25, 2005

DLLs and Reusable Code: Divide and Simplify

DLLs are a fantastic way to split up large applications, but in some situations you can still run into problems with duplicate symbols if you try to create a generic data DLL with exported global variables, using the AppGen. The solution? A hand-coded generic DLL. As Jeff Slarve shows, this is a lot easier than you might think.
Posted Friday, April 29, 2005

[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 two 600+ page softcover books: Clarion Tips & Techniques, and Clarion Databases & SQL. These books are packed with information useful to any Clarion developer. 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

Clarion Magazine

Clarion News

Search the news archive

First DerivedShell Product Released

KwikSYSTEMS(tm) ASSOCIATES has released their first DerivedShell(tm) product. DerivedShell(tm) Products can be downloaded and purchased from ClarionShop and via PayPal.
Posted Wednesday, April 27, 2005

iQ-XML 1.03

iQ-XML 1.03 is now available. There are two new functions: QualifyField allows you to further qualify a fieldname in your Local Queue with the parent for fieldname matching; AttributeFieldSet allows you to skip records added to your Queue by marking fields as REQUIRED or OPTIONAL as well as set a Clarion Format Picture. Various other enhancements included. Freeware.
Posted Wednesday, April 27, 2005

ClarionMag/Podcast Coffee Mugs!

Clarion Magazine/Planet Clarion coffee mugs are now available! You have your choice of a small mug, and/or a large mug, and/or a beer stein. The Clarion Magazine logo is on one side, the Planet Clarion logo is on the other.
Posted Tuesday, April 26, 2005

ClarionMag E-Book RSS Feed

Our latest RSS feed makes it easy to get notification of new and updated Clarion e-books. You will need an RSS reader to properly view this feed.
Posted Tuesday, April 26, 2005

New Edit-In-Place E-Book

Clarion's Edit-In-Place (EIP) capability is powerful, but difficult to master. This extensive e-book covers not just the standard EIP techniques, but also some very cool tricks with forms in place of EIP, and checkboxes for managing many-to-many relationships.
Posted Tuesday, April 26, 2005

vuAgent Updated

vuAgent has been updated with an additional Init function and some expanded capacities. Command size has been expanded from 386 bytes to 32,768 bytes (32K). Speech text length has been increased from 128 bytes to 32,768 bytes (32K).
Posted Tuesday, April 26, 2005

DynaLib 4.0.1

DynaLib 4.0.1. has been released. This new version has many new features. You can full control on practically all Clarion data structures - GROUPs, QUEUEs, FILEs and VIEWs. You can create these structures at run-time from string data. A new feature of this library allows you to create FILE structures directly from physical disk files. A new class for working with all SQL servers allows you to simply work with parametric SQL queries, getting results in queue or file. And DynaLib now supports the In-Memory Database Driver, IP Database Driver and Dynamic File Driver. The price of DynaLib Professional is $180, Dynalib Advanced is $150.
Posted Tuesday, April 26, 2005

iQ-XML Updated

An update to iQ-XML, a free XML Parser and Writer for both ABC and Clarion template chains, is now available. The biggest enhancement is added full template support for all APIs as well as Import/Export functions. Also, the Add Clarion Queue debug option has been enhanced to adjust the field types/sizes before creating the queue structure. Download is available at www.par2.com. Manually register iQXML.tpl once you install.
Posted Tuesday, April 26, 2005

New Data Ferret Website

Solace Software's Data Ferret is a Windows application which can automatically scan yellow pages type web sites and extracts the names and addresses, phone numbers and email addresses if available and places them in a file for manipulation.
Posted Tuesday, April 26, 2005

Solace Flatten Free Template

Once this template is added to the global extensions, it allows users to decide if they want their controls to appear 'Flat'. You have the option to omit buttons, checkboxes and Radio buttons. For C6 and above only. Available in the free templates section.
Posted Tuesday, April 26, 2005

EasyCOM2INC 2.02

EasyCOM2INC 2.02 is now available. Changes include: New RTL procedures, added to avoid redefining system intrinsics - ProviderType, HDC, HMenu, Picture, Dock, Owner, Step, Vartype, Project, Icon, Band; ParameterPrefix in ecom2inc.ini. gets a parameter name prefix to avoid any conflict; Renamed methods to avoid conflict with interface names; Current object shows in the progress window; Possibility to terminate generation process; Bug fixes; Template changes. Free upgrade for all registered customers.
Posted Tuesday, April 26, 2005

Easy3DStyle 2.00

Easy3DStyle 2.00 is now available. Changes include: Added new button styles: 3D boxed, Boxed, Color, VColor and HColor; Added XP Manifest support - Create XP manifest file (works in C5, C55 and C61); Added standard message function hook - new message look; Templates redesigned; More flexible settings. Price: $89 (1 license) or A bundle of Easy3DStyle and EasyResizeAndSplit for $129.
Posted Tuesday, April 26, 2005

Free Tech Books #3

A very large collection, albeit on possibly the ugliest site on the web. CapeSoft sez if you can suppress the initial gag reflex, then it's quite a good list.
Posted Tuesday, April 26, 2005

Free Tech Books #2

Free books are also available at the O'Reilly Open Books Project.
Posted Tuesday, April 26, 2005

Free Tech Books #1

CapeSoft staff point out this site with 150+ books spread across a whole range of computer related topics. It also has a good front page describing why free online books exist and so on.
Posted Tuesday, April 26, 2005

CapeSoft At South American DevCon

The 6th South American DevCon is taking place in May, and CapeSoft is proud to announce that Bruce Johnson will be attending. He will give his usually entertaining presentations on ABC and the three recently released drivers from SoftVelocity. As always, the event has been organized by Clarion users, and for the first time will be including Clarion developers from Argentina and Brazil. Previous editions of this event had 100-130 developers, with more expected this year.
Posted Tuesday, April 26, 2005

Replicate Jumps Forward

Replicate provides an automatic, driver independent, file-version independent, mechanism for replicating the data in two or more databases. Replicate underwent some major restructuring over Christmas time and the early new year. If you've been keeping up-to-date with the releases, but you're not on version 1.86, or if you're still on one of last year's versions (v1.62 or earlier) it would be a good idea to upgrade now. Some of the latest features include: LogManager ControlCenter; Optional Replicate activation made easy; Status of relating LogManagers. Cost: $399
Posted Tuesday, April 26, 2005

HotUpDates

You can use HotDates to view dates in a calendar - drill down to month, or day. Create a scheduler or weekly planner. Print your calendar straight to a printer or create a customizable date picker. Recent changesinclude:Date-Range selection (in the Year Calendar and Planner templates); DatePicker Hotkey; Sorting of child data (Planner template); Customizable zooming; and more. Cost: $239

Posted Tuesday, April 26, 2005

Secwin Online Server 1.03 Beta Released

Secwin Online Server provides immediate access to temporary or permanent product activation codes for any product which has the Secwin Online client feature enabled. Activation codes can be blocked for certain clients, products or datasets, allowing the supplier full control over product sales and distribution, but giving prospective or new clients immediate access to the product. Version 1.03 Beta now supports: Online customer registrations through your application or a product registration web page; Product activation codes can be automatically emailed to your customers. Cost: $199.
Posted Tuesday, April 26, 2005

EasyCOM2INC 2.02

EasyCOM2INC 2.02 is now available. Changes include new RTL procedures, parameter name prefixes, method renaming, bug fixes, updated templates, and more. EasyCOM2INC utility is used to automatically creating Clarion include files with the definitions of COM-interfaces from IDL file AND now comes with EasyCOM Generator and generate needed Classes. Trial version available. Price: $189.
Posted Monday, April 18, 2005

XML Parser/Writer Function Library

Robert Paresi has created an XML Parser/Writer function library to use with Clarion 6.1. Since it is a function library, you can easily attach it to Legacy and ABC applications. This function library is completely free. It comes with many functions to Load XML documents in to a Clarion Queue as well as write an XML document from a Clarion queue. Features include: Handles both Group fields as well as dimensional array fields within the queue to automatically produce child records; Ability to debug the XML document as well as display your own Clarion queue with one function call; Ability to prime and cascade queue fields when importing from an XML document; Ability to save node cursors within the document for sub-processing and quickly pop back to where you were; Documentation. Available in the downloads section.
Posted Monday, April 11, 2005

SB5 Developer Edition RC1 Build 1101

SetupBuilder 5.0 Build 1101, Release Candidate 1, (April 10, 2005) is now available.

Numerous new features, fixes, and improvements.
Posted Monday, April 11, 2005

solid software support schedule

Jens Weiermann won't be in the office for the next week and doesn't know yet if he'll have Internet access during that time, so support will, if possible at all, be somewhat delayed. Jens will be back in the office on April 18th.
Posted Monday, April 11, 2005

KwikSYSTEMS & Comsoft7 Exclusive AppShell (Duke Shells) Source

KwikSYSTEMS(tm) ASSOCIATES & Comsoft7 (as Associates) are please to announce that they have exclusively licensed the products know as Duke Shells, ands have renamed these as the AppShell product line. KwikSYSTEMS/Comsoft7 will be the exclusive sales outlet for the latest AppShell(tm) products from now on. They will also be supporting, further enhancing and adding to the existing AppShell(tm) products. There is development in the works to make some new products called DirivedShell(tm) products which will consist of existing AppShell(tm) products given a new "twist" (e.g. "Derived"). Other brand new AppShell(tm) products are also in the works.
Posted Friday, April 08, 2005

ABCFree Templates and Tools 04/06/2005

ABCFree Templates and Tools changes as of 04/06/2005 include: Added GetControlPosition method to WindowsClass; Fixed issue where ",,THREAD" was generated for threaded class declarations; Browse "Copy Button" template no longer requires name of ABC browse object.
Posted Friday, April 08, 2005

xPathManager 1.2

xPathManager 1.2 is now available. Changes include: Legacy template support added; New methods to manage paths; New code template for mapping drives; New code template code for disconnecting a maped drive; New code template for relative paths; Changed example application to show how to use Default Path feature; Two small legacy examples for Clarion 6; Updated demonstration program and installation kit for Clarion 6.1 (Build 9033), Clarion 5.5 also supported.
Posted Friday, April 08, 2005

ClarionMag Free EBook Offer Extended To Friday, April 8

All Clarion Magazine subscribers who subscribe or renew by April 8, 2005 will receive an e-coupon for one free e-book.
Posted Wednesday, April 06, 2005

ClarionMag Adds Three New E-Books

Clarion Magazine has released three new e-books: Learning The Clarion Language, Learning The Clarion Template Language, and Threading In Clarion. Non-subscriber prices are $19.95 each, subscribers $9.95. Subscribers also get free updates.
Posted Wednesday, April 06, 2005

EasyResizeAndSplit 2.08

EasyResizeAndSplit 2.08 is now available. Fixes include: GPF with the RTF control; C61 general fix; C61 iconize and restore modal window caused wrong window size and control positions. This is version for Clarion 5.0, 5.5 and 6.1 (9033). Free for all registered customers. Price: $79
Posted Tuesday, April 05, 2005

SoftVelocity Training On Demand

The first two SoftVelocity courses are ready and will start shipping out within a day or two. Check the outline of lessons that we have available so far, and I think that you will be excited at this product's usefulness to you.
Posted Tuesday, April 05, 2005

Clarion Magazine

Link To Clarion Magazine

Published 1999-02-07

Link To Clarion Magazine

If you'd like to link to Clarion Magazine, you can use any of the following images.

To download these images, right-click over the image and choose "Save Picture As" (or equivalent). When you've placed the image on your page, set the URL to http://www.clarionmag.com. Or if you wish you're welcome to link directly to the images on this server.

CLARION MAGAZINE

CLARION MAGAZINE

CLARION MAGAZINE

CLARION MAGAZINE

Using Agile Programming Techniques for the Enterprise Information System : A Case Study

Louis Coraggio – Troy State University, Florida & Western Region
Wayne A. Lundeberg – University of Phoenix, Southern Arizona Campus

ABSTRACT

The authors describe the development of an Enterprise Information System (EIS) for an ISO 9001 manufacturing firm. The system is built using rapid application development tools with the method known as extreme programming. An overview of the EIS development process, the system design goals, and a chronological narrative of EIS development are presented. Included are additional requirements and recommendations for those considering agile methods.

AGILE PROGRAMMING CONCEPTS

Information management has a cyclical history that resembles the fashion cycle of men’s ties; hold on to a tie long enough and it comes back in style. In the 1960s mainframe era, centralized processing ruled. Controlling access to resources and code efficiency were key IS goals. The introduction of the personal computer ushered in the distributed processing age. Users customized the environment and hardware became relatively cheap. Now that server based peer networks dominate the firm, centralized principles are being revisited as data integrity and controlling access are once again a priority.

In early application development focus was on getting the programming done, then worrying about documentation. The introduction of the Systems Development Life Cycle (SDLC) model and Systems Analysis and Design (SAD) methods focused attention on assessing user needs and constructing performance specifications prior to coding. The introduction of CASE development tools and object oriented programming techniques drastically reduced the expense and time for programming. The SAD methods are capital intensive, requiring a substantial investment of resources before producing a usable piece of code.

In the late 1990s, a movement toward “agile” software development gained popular momentum. The principles of the agile development are:

- Individuals and interactions over processes and tools;
- Working software over comprehensive documentation;
- Customer collaboration over contract negotiation; and
- Responding to change over following a plan. (Beck, 3)

Agile methods assume a constantly changing business and user environment. Focus is on cooperative efforts between programmers and the user community. Economic justification is the immediate benefit of the application, i.e. savings by the users will offset any increased cost for later maintenance.

Extreme Programming

The most widespread form of agile concept is Extreme Programming (XP) was first proposed by Beck (2). The formalizing of XP principles arose from the development of the

Chrysler Comprehensive Compensation system headed by Beck in 1997. Paulk (6) best summarizes the principles of XP:

1. *Planning the game* -- quickly determine the scope of the next release, combining business priorities and technical estimates.
2. *Small releases* -- put a simple system into production quickly. Release new versions on a very short cycle.
3. *Metaphor* -- guides all development with a simple, shared story of how the whole system works.
4. *Simple design* -- designed as simply as possible at any given moment.
5. *Testing* -- continually write unit tests that must run flawlessly; customers write tests to demonstrate functions are finished. "Test then code" means a failed test case is an entry criterion for writing code.
6. *Refactoring* -- restructure the system without changing behavior to remove duplication, improve communication, simplify, or add flexibility.
7. *Pair programming* -- all production code written by two programmers at one machine.
8. *Collective ownership* -- anyone can improve any code anywhere in the system at any time.
9. *Continuous integration* -- integrate and build the system frequently, every time a task is finished. Continual regression testing means no regressions in functionality as a result of changed requirements.
10. *40-hour week* -- work no more than 40 hours per week as a rule; never work overtime two weeks in a row.
11. *On-site customer* -- real, live user on the team full-time to answer questions.
12. *Coding standards* -- rules emphasizing communication throughout the code.

While neither of the authors had formally embraced the XP creed, both had an extensive history of collaborating on development of commercial applications. The authors had independently concluded that many of these principles worked. Pair programming, collective ownership, simple design, continuous integration, refactoring and coding standards had proven effective in past projects. With an on-site customer, much of the XP model was in place. The remainder of this paper describes the development of an Enterprise Information System (EIS) using (and discarding some) XP principles.

EIS DESIGN GOALS

Background

Catalina Tool & Mold (CTM) is an ISO 9001 manufacturer of precision plastic injection molds and molded plastic products. Historically, CTM has competed primarily on a quality and delivery basis. CTM is an industry leader in manufacturing technology, rapid delivery, and product design. The emergence of cheaper mold shops in the Far East has eroded the domestic market for traditionally price-sensitive customers. As a consequence, CTM focused on those customers for whom speedy delivery and/or extreme precision are primary concerns.

With annual sales of $5-$9 million and 60-90 employees, CTM was one of the larger independent tool and mold shops in the U.S. The typical job is a "one of", make to order contract, done on a fixed price basis. On average, a job is in the shop for 6 weeks with $150,000 in revenue. Repairs and modifications to prior jobs also constitute a substantial portion of sales.

In 1998, Mr. Lundeberg took over as CEO. Typical products at the time were cell phones, integrated circuit assembly trays, and industrial sprinklers. Customers pushed for faster turnaround times, experimental plastics, and closer tolerances. Investments in robotics, computer controlled machining and sophisticated CAD software could no longer be managed with the existing batch mode, report driven information system. The existing system was a patchwork of a Unix based, mainframe managerial/ financial accounting system, Excel spreadsheets, and user developed Access applications.

Much of Beck's model fit circumstances at CTM. The network platform was clearly moving to Microsoft Windows NT based servers. CAD applications that had once required specialized UNIX workstations could now be run on Intel based Windows machines. It was clear that CTM needed an information system that would allow for enterprise resource planning and paperless manufacturing using real-time data. Strategic plans also called for re-inventing the business into new markets and products.

Objectives for the EIS

Extensive discussions took place with key operation people, the Board of Directors, and financial managers. After spirited debate, the following general goals were identified.

Job Estimation **–** Many design and manufacturing jobs lost money due to misquoting the project at its inception. A quick, clean, collaborative method for quoting new work was identified as the top priority. The new estimating tools should incorporate lessons learned from prior mistakes.

Real Time Resource Monitoring and Allocation **–** In general, labor and three key processes were identified as bottlenecks. Individual operations often vary greatly from budgeted time. This was true for both engineering and manufacturing operations. Additionally, management priorities changed based on new work; customer modifications; or errors in design and machining. Managers needed real time monitoring of shop floor resources.

Preparation for ISO 9001-2000 Standards **–** To maintain certification, ISO requires that: "(t)he organization shall plan and implement the monitoring, measurement, analysis and improvement processes needed to demonstrate: … conformity of the product; …; the conformity of the quality management system; … and the continuous improvement of the quality management system." (ASQ,1) The new EIS should include monitoring and exception reporting for non-conformance as well as archiving capabilities.

Universal Visibility **–** All workstations should have at least read privileges to most major aspects of the job.

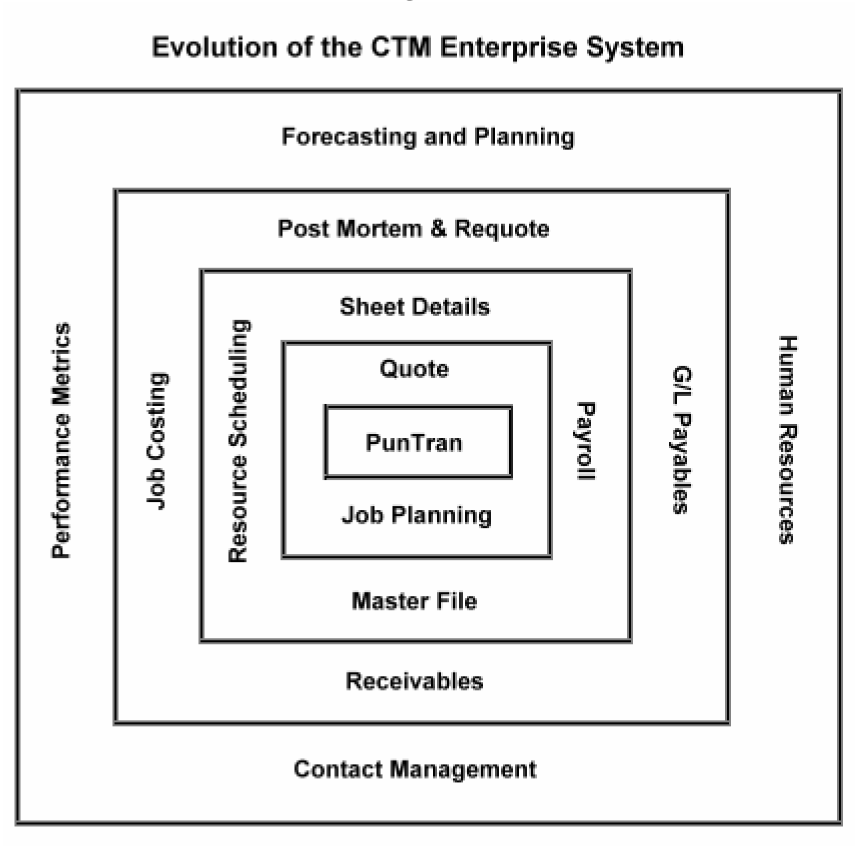
Cultural Compatibility – CTM had established cost centers, local terminology, and shop floor reporting formats. User acceptance and minimized training time were deemed mission critical.

Reduction of Redundant Entry – Much of the administrative information flow was paper to electronic to paper (e.g. time reporting.) Eliminating the initial paper step would reduce transcription errors and streamline accounting processes.

Post Production Analysis – Retrospectives on "what we did" to improve quoting skills, elevate quality and for use in performance evaluation. Analysis of the "as built" project would form the basis of a modular library of operations to be used in future job estimating.

Work began on the EIS in the summer of 1998. (Figure 1 depicts an Overview of the EIS development sequence.)

Figure 1



BUILDING THE EIS

Development Environment

The Clarion environment is a 4th generation, object oriented (OO) development environment. Applications are based on the initial construction of a relational database with file relationships, formatting, and changeable referential integrity constraints. Basic forms, queries, tables, and reports are then constructed from the database design – much like Microsoft Access. Unlike Access, Clarion allows the developer to mix and match multiple file formats in the same application and produce a stand-alone executable application that manipulates an independent set of data files. Also included is an independent report writer application that allows users to construct (and save) ad-hoc queries as formatted reports. Reports added in this manner do not require modifications of the database or recompile and reissue of applications. With a mixture of Windows 95, 98 and NT machines, Clarion's independent data and application provided maximum flexibility.

The interchangeable file formats allowed us to prototype quickly and rapidly convert live data. The proprietary "TopSpeed" file format was chosen for the final database. This format stores each table and all of its keys in a single DOS file and uses the application as a DBMS. Record hold/lock semantics, sharing, filtering and referential integrity constraints are enforced at the application level. Using this format allowed us to add, format, and manipulate individual tables without revising the entire database. It also places responsibility for maintaining database integrity and validity squarely on the developers.

The OO structure of Clarion provides an inherent consistency of formats, screens and reports. Much of the "code standard" advocated by XP is easily accomplished by customizing the templates used as a staring point for forms and tables.

Project Management for the EIS

In Clarion, the "Dictionary" construction is done independently from the "Application" development. When a Dictionary is changed, the programmer must close it, reopen the Application, and instruct the environment to integrate the changes. The Application must then be recompiled to produce the standalone "Program". Since the database is independent, each change of the Dictionary requires a conversion of the live database. With two developers, version control (and data file conversion) becomes a critical issue. To mitigate this issue, the authors created a "Master" zip disk. Only the programmer with the Master disk was allowed to change code and database fields. The official version was contained only on the master disk. Despite the availability of remote network connections, this master disk approach resulted in better personal time management and forced the developers to review and agree on changes before going live.

Each day, live data was backed up from the server to a development workstation. The live data and current versions of all Dictionaries, Applications and Programs were then archived to offline media. Each version revision was placed on a separate disk to allow the opportunity to roll back to a prior version if necessary.

While both developers worked on most aspects of the EIS, a comfortable division of labor was soon reached. As the "resident" member, Lundeberg was primarily responsible for defining business rules, final formatting of screens and reports, setting priorities, and developing program flow. Lundeberg had exclusive control of posting new releases and acted as project manager. As the "off-site" developer, Coraggio did most of the hard coding and prototyping. Database design and relationships were done together. The major benefits of paired programming were seen in developing complex database relationships and logically intensive batch updates.

User requests for changes and features rapidly got out of hand. Eventually, a formal request procedure was established to field requests. Each was reviewed by Lundeberg and typically forwarded to Coraggio for implementation in the next release. As the system grew, releases became less frequent. Many of the requested features were aesthetic or the result of disparities in local workstations. Since Lundeberg was both Project Manager and CEO, he had both the technical background and the authority to implement (or deny) changes in CTM EIS.

Phase 1 – Time Reporting

Customer pressure to reduce delivery times and price suggested the immediate need was an effective means of managing jobs in the factory. The existing Shop floor job management used a mainframe system driven by paper records of worker activity. Timecards served as both a payroll vehicle and as the primary record of cost center activity and progress. Workers typically put off filling out cards until days end, often with less than perfect recall. Transcription time added another 3-6 hours to the reporting process. As a consequence, a computer based timecard system (PunTran) was chosen as the initial system module.

The PunTran application drives most of the shop floor management. It was also the first time many shop floor workers would be asked to use PC's for cost accounting purposes. PunTran needed to be easy to use and relatively bullet proof. Workers punch in at the beginning of each shift. When punching out, net working time is calculated. Before a time record is accepted, the worker must completely allocate their time to current jobs and to cost centers. The construction of the cost center codes allowed the punch records to generate job utilization for key machine centers as well. A worker could punch in and out as often as they wished; eliminating the need to remember what he had done.

Introduction of PunTran was done formally in a company meeting. A general outline of the EIS was presented, and the initial PunTran release was demonstrated on 10 computers spread throughout the factory. A one-week period of testing was initiated. Workers entered time on both paper cards and in PunTran. At the end of the test period, worker comments were solicited.

Most found the system easy to use but slow. The prototype used list boxes and pull downs for validating choices. Most thought typing (with validation) should be an option. Workers also wanted to see a listing of their hours; to request personal time off; and to leave notes for management. All changes were implemented and the system went "live."

After a couple of weeks, behavioral issues came to light. Workers typically used one machine for punching in and another for punching out at days end. Word spread quickly that some PCs had slow clocks and others had fast clocks. In the morning, they used slow machines; in the afternoon they used the fast PC's resulting in as much as 15 minutes per day of extra time. (This was remedied by synchronizing local stations with the server clock)

Though individual workers had no edit privileges, several key administrators could fix "mistakes". It became apparent that someone was padding the records. Several security measures were considered. Eventually, transaction logging was added for any edit changes and included fields that identified the date, time, workstation, and original values. Armed with transaction record logs, the culprit was identified and confronted. Word spread that the system had “spy-ware.” After about a month, most of the initial gamesmanship and “testing” was done. PunTran became part of daily life at CTM. PunTran freed one full-time clerical worker from time card processing.

Phase 2 Estimating and Job Planning

The job-monitoring database structure is the heart of this EIS. All resource scheduling, cost accounting, ISO documentation, resource utilization, engineering designs and purchasing are tracked and allocated to the job. Determining the granularity of the job database consumed many hours of analyst time.

Integrating estimating (“quotes”) with work orders (“jobs”) needed to be as seamless as possible. The Access based estimating system (already in use) was accepted and effective. Using it as a basis, the basic screen formats, program flow and logic were duplicated in Clarion. Using the Access structure also allowed us to convert live data into the TopSpeed format. This was particularly important because of the value of using historical quote information to aid in quoting similar new projects.

Both quotes and jobs may have several “items”, each having its own budget. *(Typical quote items include tasks like product engineering, mold design, mold building and plastic part production.)* Quote items formed the basis for job items, but a job may be assembled from a variety of quotes to the same customer.

Frequently, a job requires a many:many relationship with the customer file. CTM may ship to several locations around the world, and have several different contacts for billing, project management, or technical decisions. Some jobs (like repairs or enhancements) may actually be completed before written authorization arrives. As part of job planning system, contacts within a company were spawned into a child file. Also added was a communication log that attached to each contact and to each job. This allowed project managers to track the history of engineering changes and project anomalies.

Each job consists of a parent job record and two sets of children; the job items and one record for each cost center. When a new job is created, the basic information is copied from various quotes and transferred into job items. At the same time, one child record for each cost center is created. In the quote system, individual cost centers are carried as fields in the quote record. Thus each quote field instantiates one child record in the job. Multiple items are aggregated to their individual cost center. (Appendix 1 shows a partial list of CTM Cost Centers)

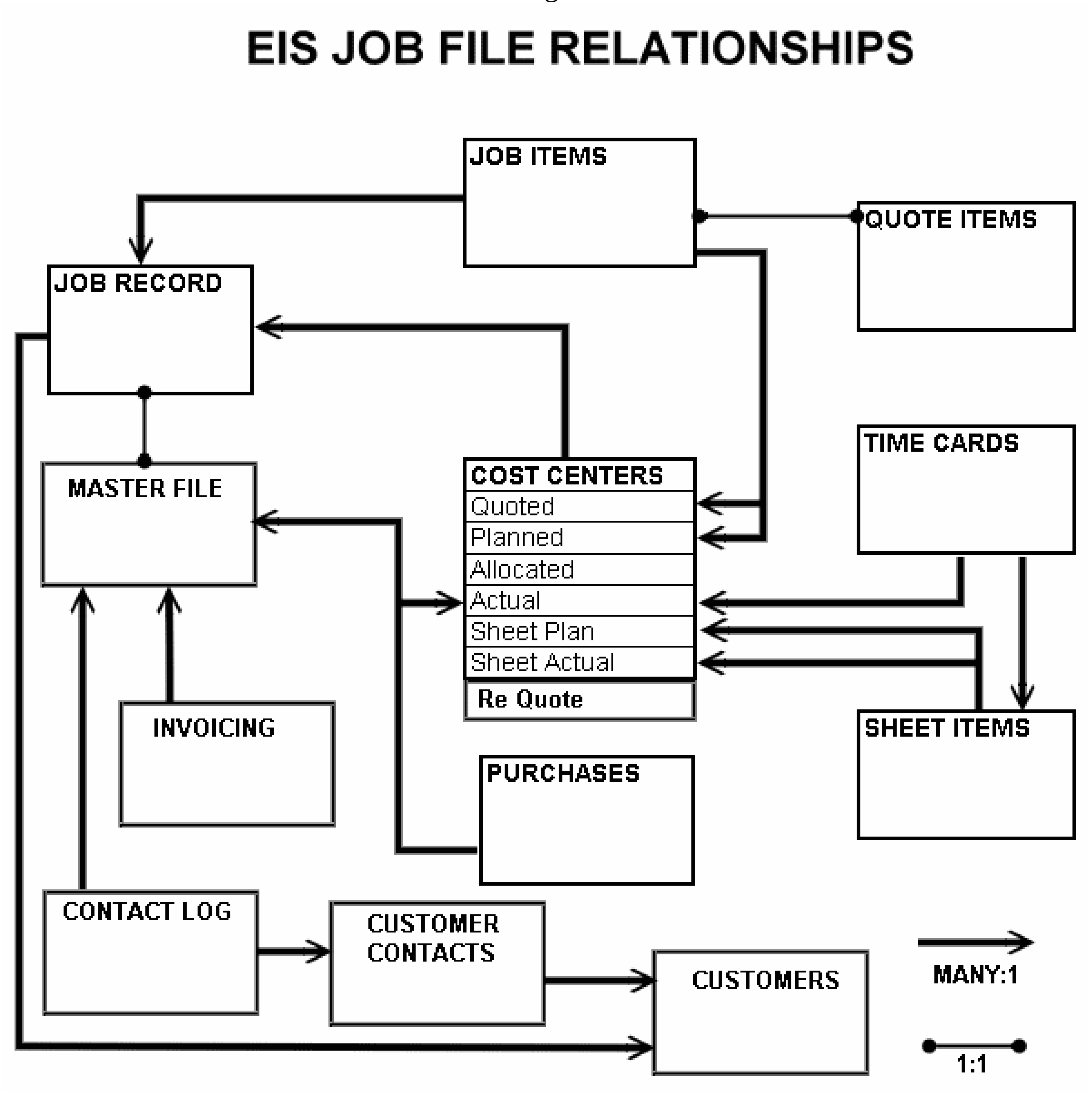
For each cost center, four fields representing the “hours” or dollars are included. There are five separate “hours” kept for each cost center activity.

- Quote – Hours directly from the quote;
- Planned – Revisions based on conditions at the time the job is created;
- Allocated – Budget hours to be publicly visible (may be revised);

- Actual – Hours tracked by the punch system; and
- Requote – Analysis of the actual time spent per process restated to account for specific process events occurring on that job. The value of this is for requoting a repeat job.

Fields for start/end dates, remaining hours and management-only "fudge" fields round out the cost center record. (Figure 2 depicts the file relationships for the primary job files)

Figure 2



Phase 3 – ISO 9001 Subsystem and Documentation

The completion of Phase 3 stabilized the database and established the central transform for the EIS. It allowed the developers to divide work independently as well. ISO 9001 compliance standards require considerable documentation of meetings, reviews, customer contacts and error correction processes. Most of these are tracked to the job level. In a normalized database, most of the ISO requirements should be part of the main job record as most requirements are 1:1. However, the sheer number of fields, and their infrequent access, led us to create a separate “master file” record for each job. This record also became the parent for most of the administrative tasks including purchasing, shipping, inventory movement and customer invoicing.

From a culture standpoint, most of the ISO documentation is after the fact. Consequently it becomes a nuisance. Much of the development work was targeted to making the job tracking record a push button operation. The CTM ISO System calls for a series of reviews and meetings throughout the project. When the master file record is created, tentative dates and participants are instantiated for these reviews. Based on the ISO plan and job characteristics, appropriate fields are displayed or hidden from users. If the program detects that a target date has past, users are prompted for an explanation or asked to fill in details.

Splitting the production and administrative sides into separate records also enhanced network performance. Typically, managers worked on the job tracking side while manufacturing floor people worked in the cost center side.

Phase 4 -- Resource Scheduling

Rolling up all budgeted and actual hours to cost centers provides a good overview of individual jobs and resource loading. However, it does not provide sufficient detail for scheduling. Since short delivery time is a major competitive advantage for CTM, scheduling is the mission critical function of the EIS. In manufacturing, tasks are identified on individual sheets of the design drawings. A prototype mold may have only one cavity; a production mold may have several cavities. The estimates, specifications and CNC programming for a single cavity might constitute a “Sheet”. For a six- cavity mold, each sheet detail would be repeated six times. Thus the sheet detail becomes the atomic unit for purposes of resource scheduling.

In a perfect world, sheet details would roll up to job items. Job items would roll up to the main job record. CTM history had proven otherwise. Competition for resources, customer changes, and rework all create a dynamically changing manufacturing floor. So another set of child records were created to accommodate shop floor scheduling. Sheet detail records are logically the children of the main job record. As cross check, sheet hours are also rolled up to their respective cost center records.

With the complex relationships and evolutionary development cycle, job records are not truly real-time. Job records are actually batch updated on-demand at the user level. Each time a user asks for details on a job, a batch process checks for changes and additions in sheet and timecard files, updates the job record in the database, then the record is displayed for that user. Since Clarion does all database management at the application level, this approach offers improved performance for all users, without forcing every entry in the PunTran system to update multiple tables.

Several versions of the resource scheduling process were modeled. Scheduling simulations were tried (based on PERT and Shortest Processing Time). While theoretically attractive, these proved impractical due to uncertainty of individual workstation times; on-the-fly redesign of the product (concurrent engineering); and resource availability. These prototypes were built with sample data, and demonstrated in focus groups with line level managers. This departure from “simple design” principles proved to be a major waste of time. This was a definite win for the XP camp.

The final version uses a list based priority system that employs a group technology approach. Resources (labor and machinery) were classified into groups based on interchangeability. Cost centers were first aggregated into groups based on similarity of skills and workstation capabilities. Each resource was assigned to one or more groups based on capabilities. For each job, tasks are also assigned to the same group designators. This allows managers to look at subsets of resources and tasks that match up. The assignment is by drag and drop. The sequencing of tasks is determined on a daily basis by the plant managers through assignment of a priority.

Phase 5 - Financial Accounting

For the financial accounting functions, a set of Clarion “templates” was purchased from another developer. These included General Ledger, Payables, Receivables, Payroll, and Purchase Orders. The authors spent about 150 hours customizing these templates. Much of the time was devoted to figuring out the template logic and data structures used by the original programmer. Importing time card records for payroll and invoicing for receivables are the primary interface with the main job based system. The authors found dealing with “foreign” code frustrating at times, the time and manpower savings from the purchase of these templates clearly justified the decision.

Payroll was the first subsystem brought on line. With PunTran in use, all of the pertinent information could be electronically accessed. Once the payroll system was customized for local taxes, direct deposits and payroll deductions, a batch process extracted payroll hours from the PunTran database to produce paychecks.

Receivables are handled within the Master File records. CTM agreements typically contain progress payments, and recurring invoices that required tie-ins to job items, quote items, and authorizations already contained within the main job system. Only summary information is exported to the General Ledger. Payables are handled within the financial system by the purchased template system. Balance Sheets, Income Statements and other standard financial reports are maintained in standard spreadsheet programs.

CTM operates using a revolving line of credit secured by capital assets and receivables. Periodic reviews of key financial ratios and performance indicators are part of the bank covenants that dictate interest rates and fees. Since CTM has a mix of recurring production and one-time projects, determining revenues, costs and work in process is a quite complex. Accurate measures of true output are performed on a monthly basis. Sales taken for a month are determined on a job-by-job basis by management. Once sales are determined, an extensive batch process rolls up all payables, invoices, time records and job progress. From this Shipment/Work In Process (WIP) report, financial ratios are calculated and the labor overhead burden is determined. Each quarter the shop labor rates (containing the overhead burden) and machine rates are reviewed and adjusted based on Ship/WIP results. The completion of the Ship/WIP process was the final link in the enterprise system. It ties the managerial accounting system to the financial accounting system and generates most of the performance metrics required by management.

Phase 6 - Marketing and Performance Evaluation

As the EIS evolved and manufacturing floor management became more streamlined, the primary focus turned to marketing. By mid 2001, the CTM client base had shifted. Jobs were smaller, with a wider customer base. CTM had opened a subsidiary operation in Queretaro, Mexico to service major manufacturers in the area. CTM marketing efforts turned toward soliciting ideas for new products from patent holders and high margin specialty manufacturers. This became known as Concept To Market (C2M).

Contact management, follow-ups, forecasting and long term monitoring of market segments became critical to the success of the C2M strategy. For the EIS, a contact management scheme was added to track personal responsibilities, dates, and prospect qualification information. Likelihood estimators for quoted jobs were added to improve sales forecasting. Much of the development effort was directed to making contact system easy to use and convenient. Reminders, and contact information buttons were tied to customer files, quotes, job progress, Rolodex and calendar functions. Wherever possible, field values were instantiated based on login, program location, and customer information. Exception reports and color-coded browse lists enable senior management to assign responsibilities and set priorities for both the corporate and individual to-do lists.

Postmortem job analysis and ISO documentation include both “as built” information and mitigation for any quality issues encountered during construction. This provided an opportunity to add individual performance appraisals for employees. These became the basis for the HR sub system. In combination with time summaries from PunTran, wage/benefits from payroll, and personal information, these became the Human Resources system. The addition of user-defined queries enables management to look at histories for individuals or groups, as well as a cross-section for any time period.

Security

While most of the EIS information is visible to all employees, there are areas limited to senior management, and accounting. Three methods are used to secure access to sensitive information, passwords; separate applications, and hidden menus. A master “switchbox” provides access to all of the individual applications. When the user logs into the switchbox with an ID and password, only those menu choices available to that user are displayed. The switchbox in turn calls individual applications and logs the date, time, station, and user in a transaction file. Individual applications also look for specific “flag files” in other folder locations to verify that they are being accessed from the CTM server. Absence of these flag files will cause an immediate shutdown.

Within an application, there are selected menus that only appear when the user executes a specific key sequence. This method is primarily used for senior management to “adjust” budgets, editing timecards, or printing sensitive reports.

The structure of the TopSpeed data files also allows the database to be copied while in use. Thus mirrored copies can be quickly generated for tracking and comparison purposes. The EIS resides entirely on a single server, with firewall protection to the outside world provided by a Microsoft Proxy Server. Source code is kept on a single local machine. While a knowledgeable employee could conceivably make copies of the data and applications, it’s unlikely that he/she could reproduce the necessary local flag files required for operation.

Refactoring Examples

As more and more of the daily activities at CTM were included in the EIS, user performance expectations also rose. Increased dependency on the EIS created higher demands on the applications. Requests for unusual reports, local use features, faster data input and different layouts dominated requests. Many of these requests were not judged to be worth the developers’ time, or implemented and never used. Though Clarion includes a “Report Writer” for ad-hoc queries, instructing users in its operation (and the structure of the database), proved impractical.

For recurring views, like receivables, export queries by example are employed. Users can tag individual fields in tables, and export the results to comma-delimited, ASCII text files. Resulting files are then imported to Excel or Access for further processing. For “one time” queries (e.g. a capital loan application), the developers created small programs to extract the data and process it with standard office software.

As the EIS grew, infrequently used reports and queries were moved into a separate application to facilitate code maintenance, and enhance performance and response-time of the core procedures. These procedures characteristically require massive sequential searches of tables, intense screen I/O, or creation of temporary files. Though this module can be called from other applications via embedded command line, it also acts as a standalone application. Thus long searches (like the Ship/WIP report) can be accomplished as a background task. Unused procedures and one-time transitional code was removed and archived.

Final Form of the EIS

When the “final” version of the EIS was released in summer of 2002 perhaps half of the U.S. mold building capacity had disappeared. CTM had survived the exodus, and won ISO 9001-2000 certification – one of the first mold shops in the world to do so. Sales for custom machining, OEM products, and C2M projects have supplemented traditional mold building.

Sales for 2001 were only 85% of 1998 levels. Over the same period, revenues per direct labor hour rose from \$53.16 to \$63.62. Much of this rise in efficiency can be attributed to the EIS. Figure 3 shows a data flow diagram of the final system. In Figure 4, summary statistics on development are provided.

Figure 3 Final EIS System Overview

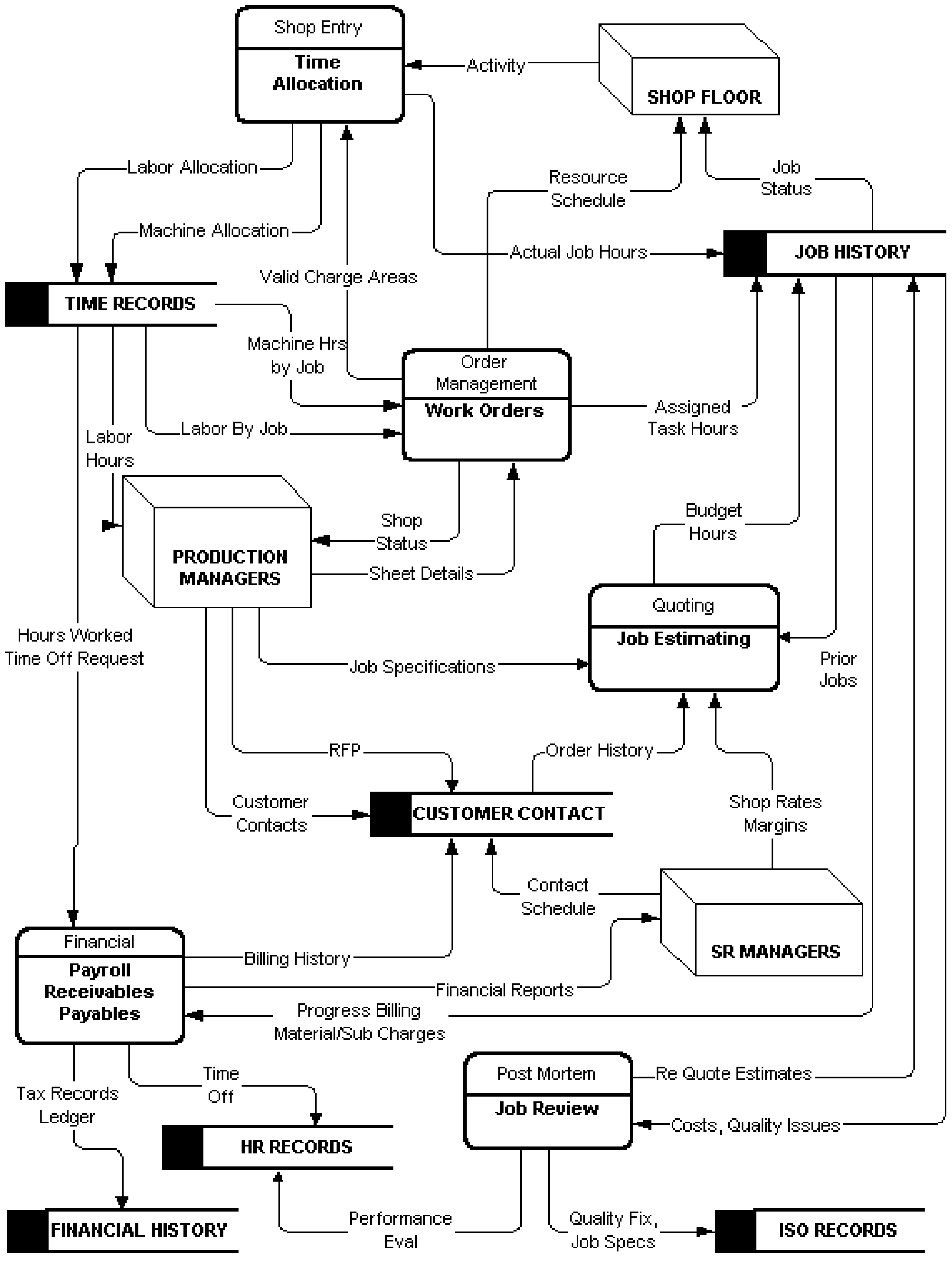


Figure 4
CTM EIS Final Statistics (June 2002)

Procedures in Financial System	275
Procedures in Main System	303
Primary Tables in Data Base	150
Externally Generated Reports	94
Estimated Programmer Hours	1400
Approximate Development Time	22 months

SO DOES XP REALLY WORK?

The CTM EIS must be considered a major success. Since the EIS development follows the XP model, the authors' answer would be a qualified yes. The biggest flaws in XP reasoning are the notions of the simple metaphor and simple coding. Stevens (7) makes a strong case for lack of planning being the major downfall in XP projects. The authors concur. Even if no formal documentation is created, time spent on the data dictionary, entity relationships and data flows are necessary planning stages. Most of the rework time was consumed in revisiting older code to add new functionality. Beck suggests that system specifications be assembled from "note cards" containing user requests. While that may be a good starting point, a trained analyst must convert those requests to fields, records and processes.

The simple coding concept ("exactly what is needed now") should be a guideline, not a rule. Adding database fields for future use, and hooks for future processing are usually viewed as major time savers. In XP style development they also serve as part of the development plan and reminders for procedures yet to be done. The inclusion of "to do" procedures also acts as a preview of coming attractions for the user community.

Paired programming has definite advantages for database development and complex logical modeling. Cockburn and Williams (4) suggest that the cost of a second programmer adds about 15% to the cost of development, offset by a 15% improvement in quality. Collaboration also ensures consistency in coding and data element naming conventions. Once the central transformation is functional (job planning), much of the development focuses on additional input and formatting output. Report writing and screen design are more modular tasks and could be delegated to additional programmers. However, the logistics of meeting, communicating, and integrating the pieces could rapidly consume any extra capacity. Having both resident and non-resident members proved valuable. Intricate and lengthy procedures are better accomplished off-site, with no interruptions. A resident developer can more effectively do testing, revisions, formatting, and refinements.

Based on the experience of Catalina's EIS the authors suggest the following additional precepts for those considering an XP approach.

The development team needs decision-making authority. – To consider all of the user requests and potential policy changes, decisions must be made quickly and have force of law. For XP development, a benevolent dictator is more efficient than a democracy.

***Analyst and programmer roles must be tightly integrated*. –** Many of the flow and logic decisions require an understanding of user perception, operations reality and programming feasibility. Adding new capabilities often creates new software performance specifications and changes in business rules. The analyst's role as user liaison is vital to acceptance.

***Anticipate transitional nulls*.** – As new tables, features and fields are added, the transition of older data must be incorporated into the code. The XP approach mandates checks for null data and some method for dealing with it. In most cases, this transitional code (or field instantiation) goes away over time. However, transition management will consume a substantial amount of coding time. The majority of incremental programming cost under XP (over SAD methods) occurs in this transition management.

Create a release plan and adhere to it. – As the system grows, incorporating changes and making the user community aware of those changes becomes a management task in itself. At CTM all requests for changes had to be submitted in writing. These were batched together and released about every two weeks. A log of changes was attached to the application, and the current release date displayed on the application title bar. New system features became part of production planning meetings.

***A motivated user community is a must*. –** The technical sophistication of the workforce and the "can-do" corporate culture at CTM is largely responsible for the success of the EIS. Direct user requests and feedback were largely responsible for many of the EIS capabilities and refinements. Without an active, receptive user community XP development will flounder.

***Save a rollback version*.** – Despite thorough testing, every new feature added does not prove successful. A complete sequential backup of applications and data will be a major time saver. It's often easier to start over with a prior version, than to undo changes.

***Be flexible in database design*. –** While the theoretical advantages of normalized database design are well documented, they may become restrictive in practice. The addition of 1:1 file relationships, extra fields, and multiple key relationships may result in major performance gains and greatly simplify development efforts.

***Allow adequate time for organization adaptation*. –** The default setting for the user community is "I don't know what I want, but I know that isn't it." Introduction of a new report or subsystem will inevitably meet resistance. During the CTM development, users were told that changes would take one to three weeks for completion. Many initial reactions disappeared after people lived with the new report for a while. The lag between activation and utilization will be longer when there is a culture change involved. Though considerable effort went into simplifying the contact management subsystem, many users simply don't use it.

Hold on to back up systems as long as possible. – Some bugs may take weeks or months to surface. Issues with triggered events (like year end closings or quarterly taxes) may reveal problems that were not caught during routine operations or pre release testing. Trial closings with backup data are highly recommended.

***Consider batch, on-demand processing*. –** Many reports are infrequently needed, or are limited to very few users. Using spawned, batch-style processes, (with locally generated files), for these reports will save extensive rework of database keys and relationships. Batch processes are typically easier to modify for transitional nulls.

***Limit end user computing*. –** While major development is going on, users will insist that provided reports and screens don't meet their needs. Some will create shadow systems without recognizing changes in business rules and the effects of transitional nulls on data integrity. If the need for ad hoc figures arises, the development team should provide it. Agile development will exacerbate reconciliation problems.

***Consider user documentation issues*. –** Most programmers and analysts despise doing user documentation. The accelerated pace of XP methods exaggerates this problem. At CTM several key users act as testers and become "Local Resident Experts" (LRE.) While formal training was regularly scheduled, the LREs provided the majority of instruction. In commercial, "shrink wrap" situations, this approach may not be feasible.

References

1. American Society for Quality, *American National Standard Quality Management Systems – Requirements for ISO 9001-2000*, December 13, 2000, Quality Press, Milwaukee, WI

2. Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, MA, 1999

3. Beck, K. et.al., *Manifesto for Agile Software Development*, http://www.agilemanifesto.org/, Confirmed October, 2002.

4. Cockburn,A., Williams, L., *The Costs and Benefits of Pair Programming*, XP2002, http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF

5. Hoffer, J.A., George, J.F., Valacich, J.S., *Modern Systems Analysis and Design, 3rd Edition*, Prentice Hall, Reading, 2001.

6. Paulk, M.C., *Extreme Programming from a CMM Perspective*, IEEE Software, Vol. 18, No. 6, November/December 2001, pp. 19-26.

7. Stephens, M., *The Case Against Extreme Programming*, http://www.softwarereality.com/lifecycle/xp/case_against_xp.jsp, Feb 3,2002

Appendix 1 -- PARTIAL LIST OF CTM COST CENTERS

	General Labor		Machine Centers		Materials
1	Layout	141	CNC Electrodes	201	Mold Base
2	Detail	150	EDM Sinker	202	Steel/MB Plates
3	Checking	151	Mold Base CNC	203	Pins/Comp
4	Electrode Design	152	CNC Machining	204	Graph/Wire
5	Programming	157	Mold Sample	205	Hot Runner Sys
6	Training	158	Production Molding	206	Misc
7	Run Prints	160	Wire Steel	207	Tools
8	Database Processing	165	Wire Electrodes	208	Resin
11	CNC MB				
12	Conventional MB				
21	CNC Machining				
22	Conventional Machining				
23	Grinding		**Administrative**		**Subcontracting**
41	CNC Electrodes	506	Inspection Admin	301	Heat Treat
42	Conventional Electrodes	510	Engineer Admin	302	Thr. OD Grind
50	EDM Sinker	511	Indirect	303	Turning
60	Wire Steel	512	Supervision	304	Jig Grinding
65	Wire Electrodes	514	Meeting/Training	305	Plating
70	Honing	517	Shop PLT	306	Gundrilling
75	Heat Treating	527	Project Management	307	Polish
80	Polishing	618	Admin Salary	308	Mold Design
90	Assemble	622	Admin Holiday	309	Text Engrave
91	Meeting	623	Admin PLT	310	Sample
111	Production Molding	624	Admin PTO	311	Freight
112	Mold Sample	217	Personal Time Off	312	Misc Subcontract
120	Inspection	218	Shop Holiday	313	Wire - EDM
121	First Article Inspection			314	Machining

Clarion Magazine

Limit An App To A Single Instance: DDE Strikes Back

by Steven Parker

Published 2005-04-12

In days of old, when programmers were bold and DDE was the ultimate tool … if one wished to limit one's application to a single instance, it was quite easy. But Microsoft has deprecated DDE, and has done its best to move programmers to other solutions. As a result, that old DDE instance-limiting code doesn't always work as expected. Unless you add a modern twist.

Limiting with DDE

Here's how the original DDE approach works. Following the lead of a Richard Taylor news group posting in October 1999: "The trick here is to make your program a DDE Server and have it look for an already running instance of itself before fully loading."

To implement this, Richard recommends, first, declaring a global variable:

```
Channel LONG
```

Second, ensure that the Clarion prototypes for DDE are available. In the Inside the Global Map embed:

```
Include('DDE.CLW'),ONCE
```

Also, prototype a Windows API call, to bring an app to the foreground, in the same (Global Map) embed:

```
  MODULE('Windows API Call')    BringWindowToTop(LONG),BYTE,RAW,PASCAL,|
NAME('BringWindowToTop'),PROC
  END
```

Next, in the Program Setup embed point, test whether there is a previous instance:

```
Channel = DDECLIENT('MYPROGRAM')  ! look for running instances
    ! of this program
IF Channel <> 0                              ! and if found
  DDEEXECUTE(Channel,'INFRONT') ! tell the running instance to
    ! take focus
  DDECLOSE(Channel)
  RETURN                        ! then get out
END
```

In the Frame's `Init` method, after opening window, set up a DDE server:

```
    Channel = DDESERVER('MYPROGRAM') !set program as a DDE Server
```

Finally, in the Frame's `ThisWindow.TakeEvent` method, at the top of the `CYCLE`/`BREAK` support embed point:

```
    IF EVENT() = Event:DDEexecute
      IF DDEVALUE() = 'INFRONT'  !tells it to bring itself to the
        !front
        AppFrame{PROP:Iconize} = FALSE
        BringWindowToTop(AppFrame{PROP:Handle}) !and this does it
      END
    END
```

What happens is this: When the app starts, it tries to set itself up as a `DDECLIENT`. `DDECLIENT` requires that a (named) `DDESERVER` already exists. In this case, if `DDECLIENT` returns a good value, the app must already be running.

If this is the case, a command is sent to the server. The server, listening for DDE events, responds to the DDE command by coming to the top.

The app attempting to open then terminates.

If `DDECLIENT` fails, the app must not be running; it couldn't attach to the server. In that case, it registers itself as a `DDESERVER`. This is what other attempts to start the app try to find.

Microsoft has its say

This code worked when DDE was Microsoft's latest salvation. Now, Microsoft wants to move programmers away from DDE and they have "deprecated" (read "crippled") it, so that DDE calls are much slower. But this code still works and if one's users have a modicum of patience, it works perfectly.

However, if the app does not appear in a timeframe acceptable to the user (i.e., instantly) and that user tries to start the app again, the new-found slowness of DDE may well allow multiple instance of the app to start.

In A Naïve Look at the Mutex, I stated that a mutex could be used to ensure that only a single instance is running, as follows:

```
    Limiter &= NewMutex('thisApp')
    If Limiter &= NULL      !very serious problem
      Message('Error: Mutex cannot be created')
      Return
    Else                  !try for half second
      Result = Limiter.TryWait(500)
      If Result <= WAIT:Ok !program not already running, continue
        ProgramStarted = True
      Elsif Result = WAIT:TIMEOUT !program is already running, +
                         !    warn user
        Message('Another instance of this program is running. ' &|
                'Please switch to that instance.','Don''t Do ' &|
                'That!',ICON:Hand) !and terminate
        Return
      Else                    !another  very serious problem
        Message('Wait failed')
        Return
      End
    End
```

A mutex is not only a synchronization object, it is a Windows kernel object. Therefore, it should be more than fast enough.

This code works by attempting to create a named mutex. If this attempt fails (`Limiter &= NULL`), something very serious is wrong. The app is terminated.

Otherwise, it tries to get the mutex. If it can, the app is not running and initialization can continue. If the mutex already exists (`Result = WAIT:TIMEOUT`), the app is already running.

All that is missing is notifying the running copy and bringing it to the top.

To effect this, *all* of the "old" DDE code is required. The code that actually notifies a running instance simply has to be moved into the mutex test:

After Program Code statement:

```
    sLimiter &= NewMutex('MYPROGRAM')
    If sLimiter &= NULL
      Halt(,'This wonderful program encountered a fatal ' &|
            'problem starting up.')
    Else                    ! try for 1 second
      Result = sLimiter.TryWait(1000)
      If Result <= Wait:OK
        !Program started
      Elsif Result = Wait:Timeout
```

```
        Channel = DDEClient('MYPROGRAM')
        If Channel <> 0
          DDEExecute(Channel,'INFRONT')
          DDEClose(Channel)
          Return
        End
      Else
        Halt(,'System problem trying to open configuration files.')
      End
    End
```

I do get occasional reports of the app not starting but multiple instance showing in Windows' Task Manager. This seems to imply that it is hanging at the attempt to get the mutex. Perhaps I have embedded my code a bit too early. But, I do *not* get multiple instances of the app running.

Allowing multiple instances, with control

I was recently asked to allow a specific user to run two instances of the app. All rational suggestions were rejected; they wanted two different icons on the desktop with each instance pointed to different data set (actually, different file servers, too). And I wasn't going to do a one-off version or a serial number check ….

But. A unique name is necessary for both the mutex and the DDEServer. If the customer is willing to run each instance from a separate directory, the Clarion `Command()` statement guarantees the uniqueness needed to provide multiple instance with some semblance of control.

The code now becomes:

```
    sCommandName = Command('0')
    sLimiter &= NewMutex(sCommandName)
    If sLimiter &= NULL
      Halt(,'This wonderful program encountered a fatal problem ' &|
            'starting up.')
    Else
      Result = sLimiter.TryWait(1000)
      If Result <= Wait:OK
        !Program started
      Elsif Result = Wait:Timeout
        Channel = DDEClient(sCommandName)
        If Channel <> 0
          DDEExecute(Channel,'INFRONT')
          DDEClose(Channel)
          Return
        End
      Else
        Halt(,'System problem trying to open configuration files.')
      End
    End
```

The only problem is that *any* attempt to start the app fails with a message.

I checked. `Command('0')` returned the correct fully qualified path and program name, just as if I had typed it in a DOS box.

It was suggested that I try the optional error parameter of the `NewMutex` statement, which would make the problem obvious:

> **`NewMutex (name, owner, <error>)`**
>
> **`name`** A string constant or variable that names the new IMutex object. If a name parameter is not supplied, NewMutex is used for synchronizing threads only. If a name is supplied, NewMutex can be used to synchronize multiple processes rather than just the threads within a process.
>
> **`owner`** A BYTE variable or constant that specifies the initial owner of the Mutex. If this value is TRUE, and the caller creates the Mutex, the calling thread obtains ownership of the Mutex. This is the equivalent of calling Mutex.Wait immediately after NewMutex(). If the caller did not create the mutex, then the calling thread does not obtain ownership of the Mutex. To determine if the caller created the Mutex, you need to check the value of Err
>
> **`error`** A LONG variable or constant that returns any operating system error. ERROR_ALREADY_EXISTS (183) indicates that the caller did not create the Mutex (a handle is still returned but owner is ignored) because another process or thread has already created the Mutex.

Unfortunately, this returned the totally unhelpful error: Path not Found. Path not found indeed!

I do not remember who it was who solved this dilemma, for surely I owe him a beer, but it turns out that the Windows kernel is not fond of back slashes (shades of UNIX!). Replacing back slashes with forward slashes solved the problem:

```
    sCommandName = Command('0')
    Loop i# = 1 to Len(Clip(sCommandName))
      If sCommandName[i#] = '\'
        sCommandName[i#] = '/'
      End
    End
    sLimiter &= NewMutex(sCommandName)
    ! etc.
```

True, there are synchronization objects that allow multiple waits. But, (1) I haven't mastered them and (2) I don't recall them being testable.

So, there you have it. Despite Microsoft's best efforts (to date, at least, and until they further muck with the DDE code base), you can still use DDE, with a steroidal boost, to limit an app to a single instance.

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 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

Version Control with CVS and Clarion 6.x

by Bernard Grosperrin

Published 2005-04-15

Version Control is a very wide subject, and I am not going to cover the whole thing from A to Z in a single paper. Nardus Swanevelder did an excellent job already, presenting the basics of Concurrent Versions System (CVS) and how you can use it.

I am going to look at how you can use CVS to keep track of SoftVelocity releases and hotfixes, as well as of third party tools, and how you can keep track of changes you made to one of the releases of your software, after you've started development of the next wonderful killer version.

So, in a way, I am going to look at CVS starting by the end, the results, then look at how to organize your daily work with Clarion and CVS. Nardus covered the installation on a standalone machine; I will briefly explain the installation of CVNST on a Linux server, but this is not going to be a "How-to" install CVSNT article. I will refer you to web pages doing this much better and in greater detail.

You will also discover that CVS use is not limited to keeping track of your development, it can also be used for everything else you do where keeping revisions of text files can be important, like a web site, documents, SQL scripts, installation scripts, etc, etc.

Let me quickly state which software I'm using, so that you can find and install those tools:

Server

For the server, you can use either CVS or CVSNT. Both can be installed on Linux, CVSNT has a Windows version.

CVS: https://www.cvshome.org/

CVSNT: http://www.cvsnt.com/cvspro/ and http://www.cvsnt.org/wiki

Client

There are a number of clients out there for Windows, Tortoise being interesting and easy to use, but there are many features in WinCVS missing in Tortoise, one of them being the ability to extend the commands via Python Macros, which I will do to customize WinCVS for Clarion.

WinCVS: http://www.wincvs.org/

Python

It may seem strange that an article on version control introduces a scripting language, but WinCVS uses Python as its macro language, and I would suggest you download and install Python (version 2.3 or 2.4), along with the PythonWin extensions, for which I will have a use later...

Python: http://www.python.org/

PythonWin: http://www.python.org/windows/pythonwin/

Differences Comparison tool

One of the strong points of version control is the ability to compare different revisions of a file. CVS has a built in tool, but it is not too user friendly as it compares line by line, out of context. There are other, better tools. Personally, I use Beyond Compare, but do your own search and select your own.

Beyond Compare: http://www.scootersoftware.com/

Editor

You can set up WinCVS so that it will open a file in your favorite programming editor. I use Multi-Edit, but you can use any editor you like. Multi-Edit can also be set up to work with CVS, so if you work quite a bit on classes, SQL Scripts, HTML, CSS, etc. you can do all your work and version control management without leaving Multi-Edit!

Multi-Edit: http://www.multiedit.com/

Keeping Track of SoftVelocity versions and hotfixes

Now, let's assume you have a CVS server installed/available somewhere, and you have installed WinCVS (although what I describe here will work just as well on a standalone system as described by Nardus).

Say I work with Clarion 6.x, and it happens that SoftVelocity just released a new hotfix. But I have made some changes to ABFILE.CLW, because I wanted my own version of PrimeAutoIncServer (and there are too many privates properties to derive cleanly), or I changed some templates, or none of the above, as I never modify anything directly, I just want to be able to roll back easily if something is wrong with the new fix.

Let's see how CVS can help.

The first thing to do, if not already done, is to put my Clarion6 directory under version control. I could either do that for the whole Clarion 6 directory tree, or select only the directories I want. My directories 3rdparty,Bin,Lib, Libsrc, and Templates are under version control. I will explain later how you do that; for now, let's just say it's done.

I just downloaded Clarion 6.1 hotfix build 9031, and I want to update my Clarion 6 working directory.

As SV hotfixes check that I have the correct versions of the DLL before installing, I have to install on an existing Clarion directory, but *not* my working directory. Let me be clear about this: You need to have a Clarion directory tree, which you will use only to "feed" CVS. Your "real" Clarion working directory tree is *only updated* from CVS, never directly from SoftVelocity updates. To do that, I simply copy my whole Clarion 6 directory, renamed with the last hotfix number: something like Clarion6_9031. The cleanest way to proceed would be to install a clean Clarion6_updates directory from the original CD, then update it with each of the releases/hotfix, and each time do the procedure I am going to describe. That way you would have all SoftVelocity releases under version control. If you did not make any modifications to SV shipped files, I would recommend this as the best start.

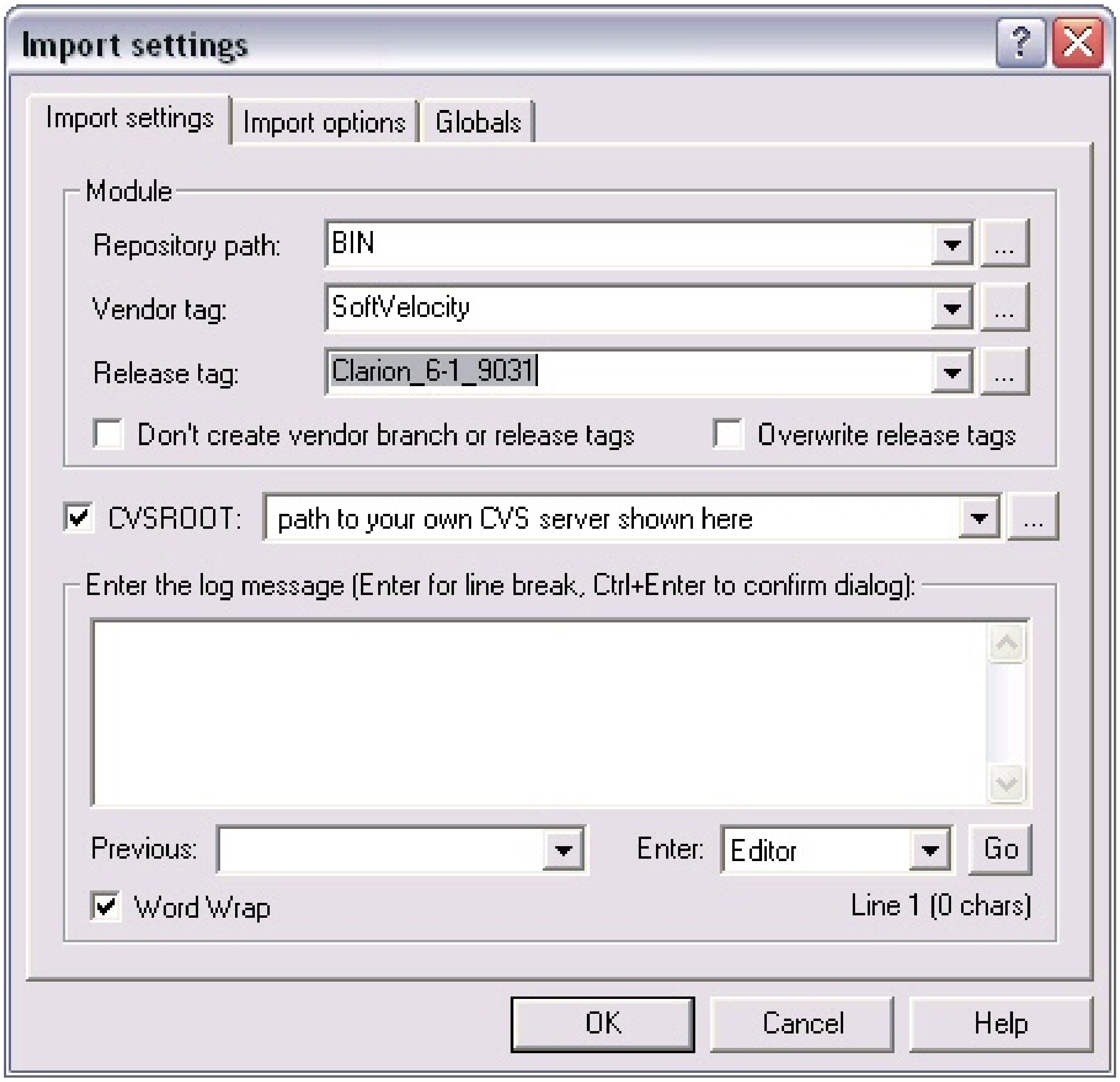
Now that I have a Clarion6_9031 directory, I need to tell version control that I am importing a "vendor update". I select the first directory from my "vendor update directory tree" that I want to have under version control, BIN, right click and select "import Module" in the popup menu. It is important to check that the directory name matches exactly the directory name existing on the server, as I want this new import to go to the *same* directory on the server where my

Clarion tree is. Only my local directory, to import from, must be different. WinCVS run an import filter, to differentiate between text files and binary files. There is normally no correction to be made here, so I click OK to go to the Import Settings Window.

What I am interested in, in this window, as seen in Figure 1, are the Vendor tag and Release tag settings. Vendor is SoftVelocity, release is Clarion_6-1_9031. Note that you can't have spaces in a tag name. Note also that, my server being a Linux box, I have to be careful with cases. For instance, if on your server the directory is Bin, rename the directories on your "vendor update" directory tree as created by Clarion's install to match exactly; if you import a BIN directory, you will create another directory, not update the existing one!

Once you click on OK, CVS will update your BIN directory *on the server* with SV's latest patch.

Do the same for each directory under version control.


Import settings
Import settings
Import options
Globals
Module
...
Enter the log message (Enter for line break, Ctrl+Enter to confirm dialog):
Go
OK
Cancel
Help

Repository path: BIN

Vendor tag: SoftVelocity

Release tag: Clarion_6-1_9031

☐ Don't create vendor branch or release tags ☐ Overwrite release tags

☑ CVSROOT: path to your own CVS server shown here

Previous:

Enter: Editor

☑ Word Wrap

Line 1 (0 chars)

Figure 1. CVS vendor Import.

Once you are done importing Modules for this new release, select your Clarion 6 working directory from WinCVS, and select Update for each directory under version control. Now, if you think this is a slightly tortuous procedure to update Clarion, and take 30 minutes where it took usually less than five, I will entirely agree with you! So, where are the benefits?

They are almost too numerous to list.

First, it's now pretty easy to see what has changed between two revisions. For example, I can see that AbFile.clw (Figure 2)did not change at all between the last three hotfixes, while AbDrops.clw (Figure 2) has a change between 9029 and 9030. How do I know this? I just right click on the files, and select Graph from the popup menu:

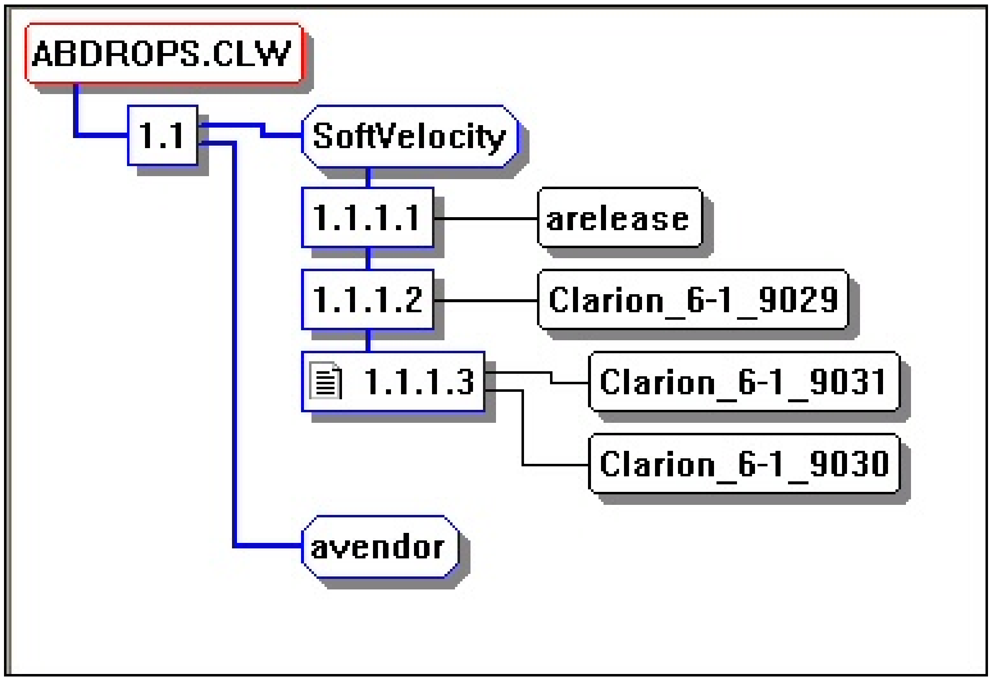
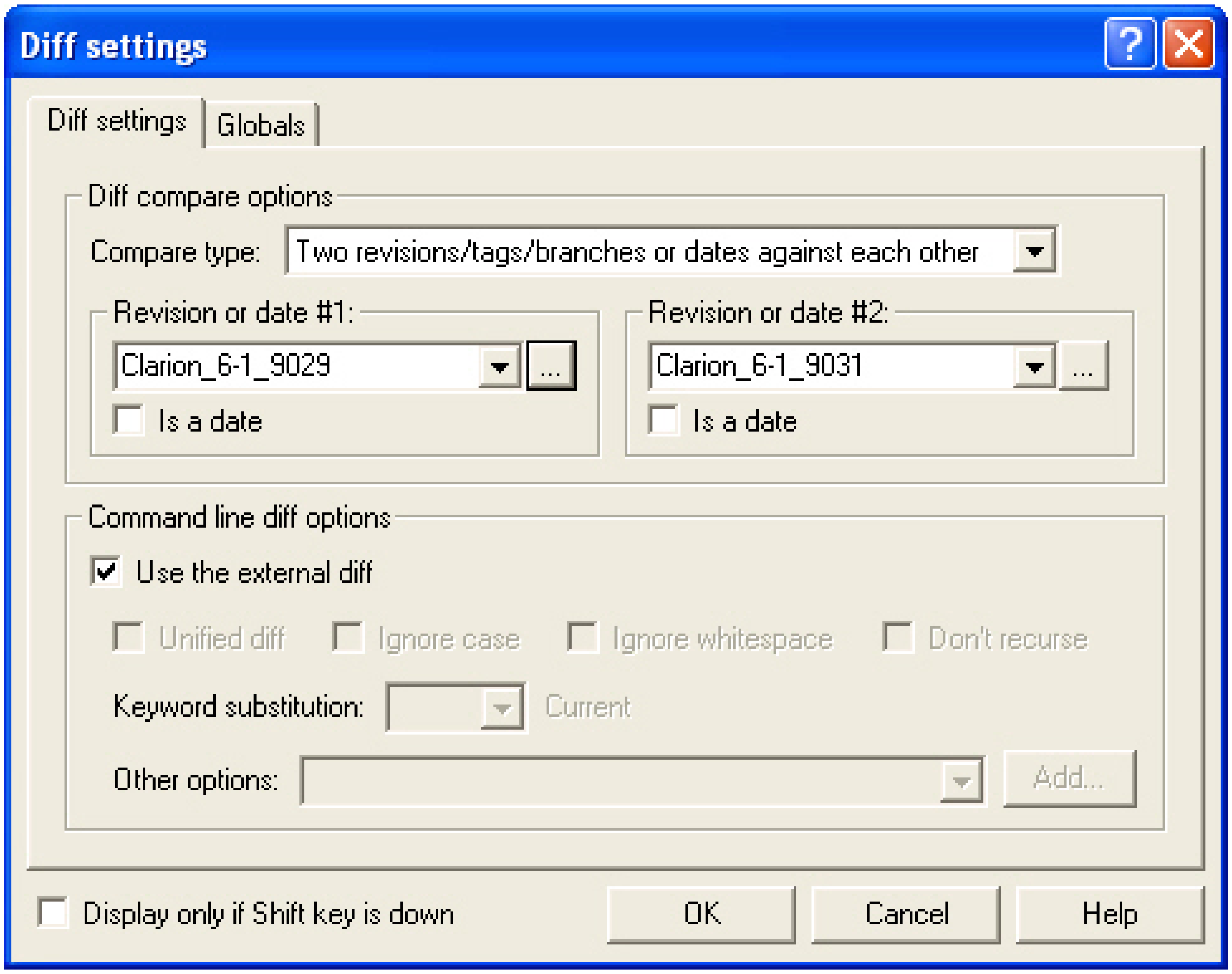


Figure2. AbFile and AbDrops graphs

Verrsion 1.1 is the original version, created when I imported the whole module into CVS (I was on hotfix 9028). You can see that for ABFile, vendor version 1.1.1.2 has all three release labels pointing to it, indicating it's the same version and there has been no changes, while Abdrops has only 9029 pointing to revision 1.1.1.2, and 9030 and 9031 pointing to revision 1.1.1.3. I can as easily look at those changes. With AbDrops.clw selected, I select Diff from the popup menu. In the dropdown list box on top "Diff compare options", select "Two revisions/tags/branches or dates against each other", as seen on Figure 3, Diff settings..



Diff settings

Diff settings | Globals

Diff compare options

Compare type: Two revisions/tags/branches or dates against each other

Revision or date #1: Clarion_6-1_9029

[ ] Is a date

Revision or date #2: Clarion_6-1_9031

[ ] Is a date

Command line diff options

[✓] Use the external diff

[ ] Unified diff [ ] Ignore case [ ] Ignore whitespace [ ] Don't recurse

Keyword substitution: Current

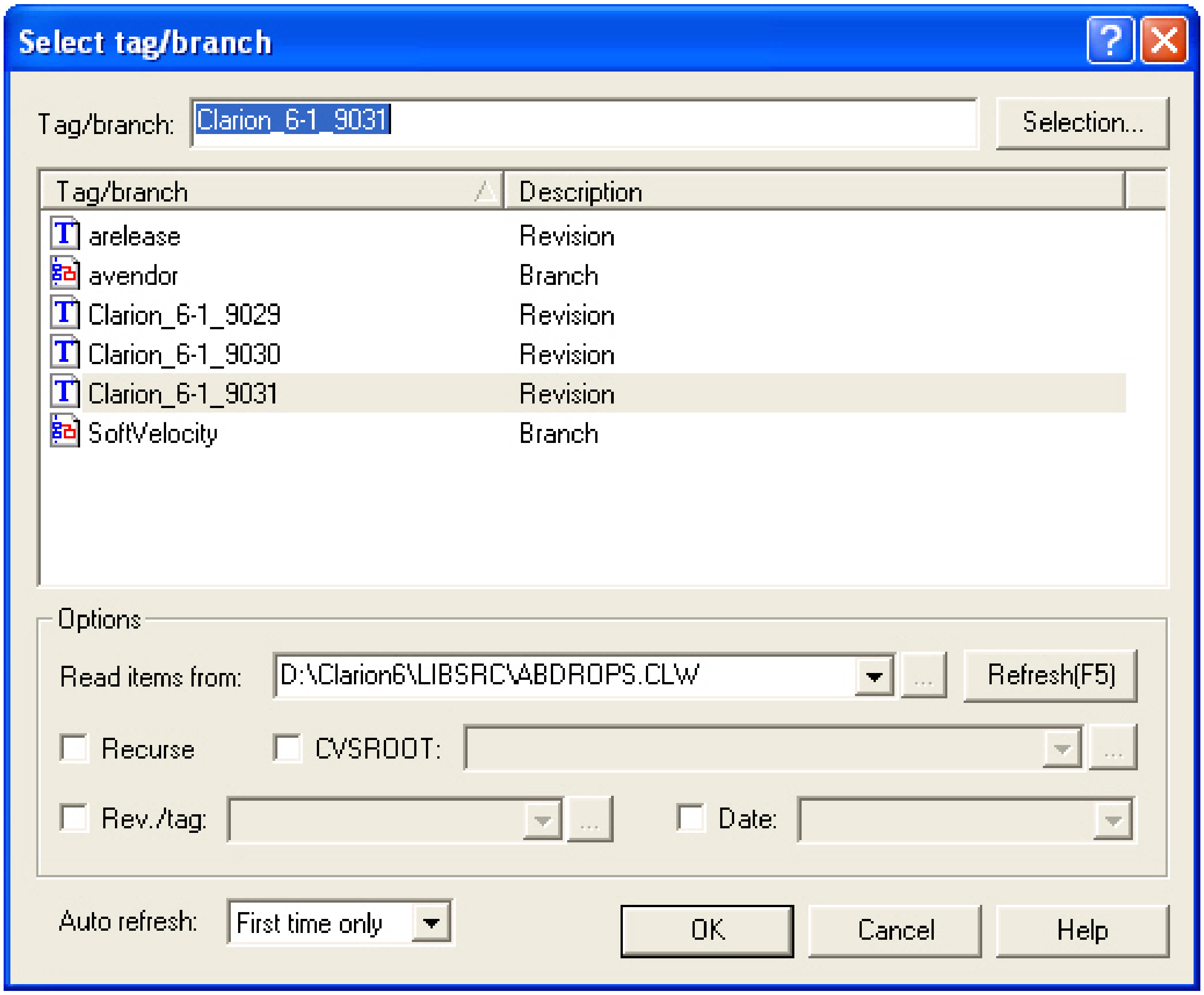
Other options:

[ ] Display only if Shift key is down

Figure 3. Diff settings.

For revision or date #1, click on the lookup button to have "select tag/branches" displayed (Figure 4), select the older version, then select the newer for Revision or date #2.

Don't forget the check the box "Use the external diff" if you want to see the changes in your preferred compare program. I use Beyond Compare.



Tag/branch: Clarion_6-1_9031

Tag/branch	Description
arelease	Revision
avendor	Branch
Clarion_6-1_9029	Revision
Clarion_6-1_9030	Revision
Clarion_6-1_9031	Revision
SoftVelocity	Branch

Options

Read items from: D:\Clarion6\LIBSRC\ABDROPS.CLW

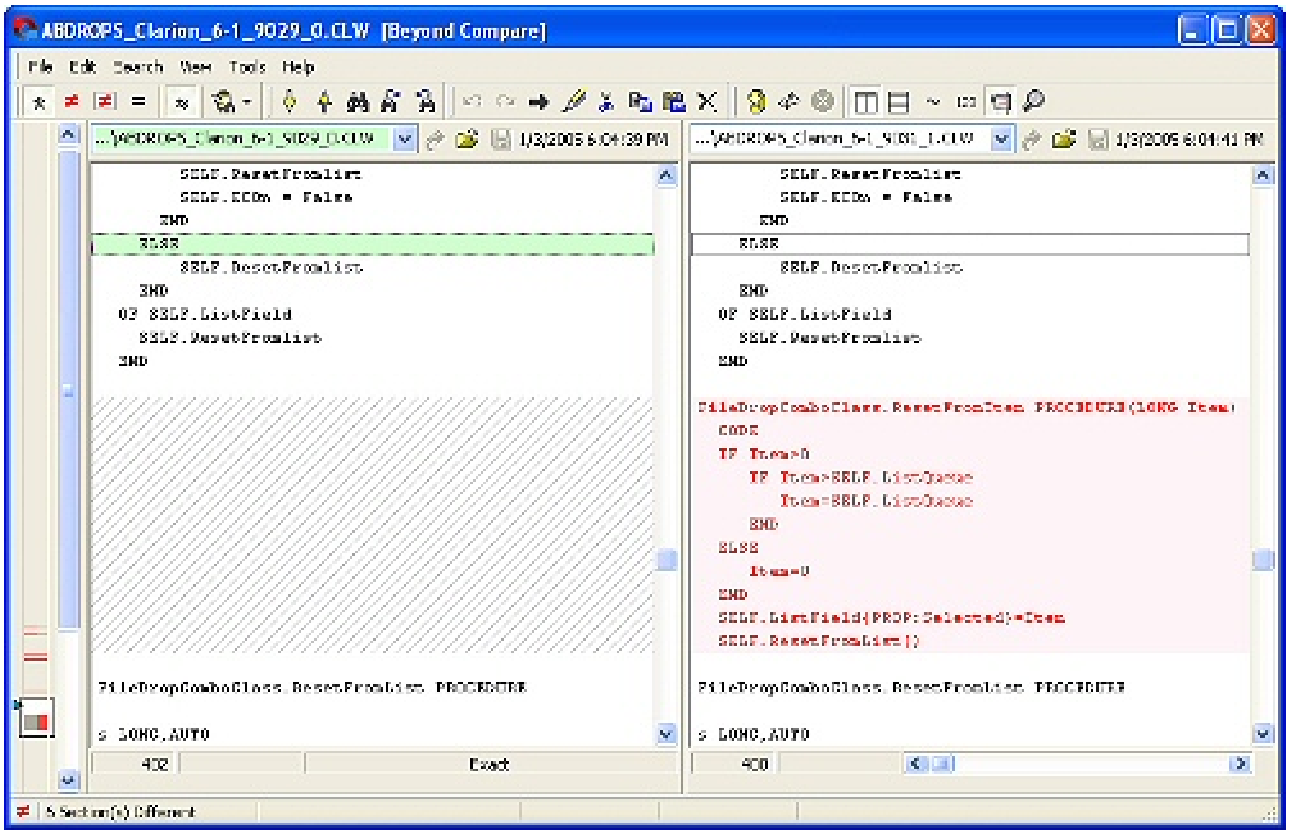
☐ Recurse ☐ CVSROOT:

☐ Rev./tag: ☐ Date:

Auto refresh: First time only

Figure 4. Select a tag/branch

Among a few other lines changed (Beyond Compare shows six sections different, Figure 5), I can discover that 9030 and 9031 have a new `ResetfromItem` method in the `FileDropComboClass`. I could have discovered the same thing reading SV release notes item (FEATURE: New method to reset the FDCB to an item in the list. `ResetFromItem PROCEDURE(LONG Item)`), but that does not tell me which file is involved and how the code looks like, and where would be the fun, anyway?



ABDROPS_Clarion_6-1_9029_0.CLW [Beyond Compare]

File Edit Search View Tools Help

...\ABDROPS_Clarion_6-1_9029_0.CLW 1/3/2005 6:04:39 PM

```
        SELF.ResetFromList
        SELF.ECOn = False
      END
    ELSE
        SELF.ResetFromList
    END
  OF SELF.ListField
    SELF.ResetFromList
  END


FileDropComboClass.ResetFromList PROCEDURE

s LONG,AUTO
```

402 Exact

...\ABDROPS_Clarion_6-1_9031_1.CLW 1/3/2005 6:01:41 PM

```
        SELF.ResetFromList
        SELF.ECOn = False
      END
    ELSE
        SELF.ResetFromList
    END
  OF SELF.ListField
    SELF.ResetFromList
  END

FileDropComboClass.ResetFromItem PROCEDURE(LONG Item)
  CODE
  IF Item>0
    IF Item>SELF.ListQueue
      Item=SELF.ListQueue
    END
  ELSE
    Item=0
  END
  SELF.ListField{PROP:Selected}=Item
  SELF.ResetFromList()

FileDropComboClass.ResetFromList PROCEDURE

s LONG,AUTO
```

400

5 Section(s) Different

Figure 5. Comparison with Beyond Compare (view full sized image)

Tagging a product release (version)

One of the most recurrent problem of any software company, either individual or corporation, is to be able to freeze a development environment at a specific point in time, matching the release of a product.

Let's say you proudly release version 10.09 of your killer application, and immediately start working on the next, the BIG one, version 11, which will have all the features you have been dreaming to implement for so long. You conscientiously update your third party products, as well as your Clarion version, change a few templates, and work, work, work....

Suddenly, the phone rings. Your biggest, most important client has discovered a showstopper bug, while you are still months away from releasing version 11. You know you have to immediately stop any new development to fix that bug. And then, it hits you. You *can't!*

You can't because you have no ideas what version of each third party tool you were on. When you loaded the APP you have so preciously backed up and labeled Killer App 10.09, you get tons of error messages, templates missing, prototypes of functions and classes changed, etc, etc.....

OK, maybe that's a bit of a caricature, as it's possible to backup the entire Clarion directory each time you have a new release, but, I don't do that systematically enough, and I know that this is not purely a fairy story…

So, how do you use CVS to avoid this kind of Clarion developer drama?

Modules

To manage complete versions of your development environment you have to define a module on your server.

CVS use the word "module" for any directory in your server repository. Here, we are talking about a logical, or virtual, directory, not a physical directory.

So, what is a logical module, seen from the server? It is a way to group files and directories belonging to a project. In fact, it's a pretty simple ASCII file, named, guess what, "modules", residing in your server's CVSROOT directory (which was automatically created when you first initialized the CVS server, or your local installation).

I would suggest, if you have not already done so, that you Checkout CVSROOT somewhere on your local drive, so that you can edit your version control configurations files, and commit them, without sitting in front of your server. You will even be able to go back if you make some mistakes, as this a way for CVS to put itself under version control! (Please, please, if you are on a standalone machine, *never* directly update files in the repository!)

To demonstrate a module, I will use the Clarion example Invoice application. But to make the example more realistic, I will move the directory to an Invoice directory on my drive. What I want to have in this Module for any Invoice product release is everything that I might need to have later on to be able to change and recompile. This means I need to include the directories and files required by the application, such as any Clarion or third party BIN, LIBSRC, TEMPLATE, and IMAGE directories, as well as my working directory. I would need to add to that any other file/directory used in my final product, like the install script, SQL scripts if any are used, etc, etc.

To put my new folder Invoice under version control, I select it from WinCVS. I then right-click on it, select Import Module from the popup menu; the import filter window show what files will be imported as binaries, and which one as text. As for now I am not digging into Clarion 6's use of CVS, I leave aside APVs and DCVs, the renamed TXAs and TXDs generated by Clarion for Version Control. I will let CVS handle APP and DCT files, which it recognizes as binaries. (I will look at Clarion 6 integration with CVS later, but this will show to you that, even if you have an older version of Clarion, you can benefit from CVS, although not with the maximum flexibility).

Here is what the graph for Invoice.App looks like after initial import (figure 7). For an initial import from WinCVS and with CVSNT as server, you need to select "Don't create vendor branch or release tag, on the tab import settings, and "Create CVS directories while importing" on the Import options tab.

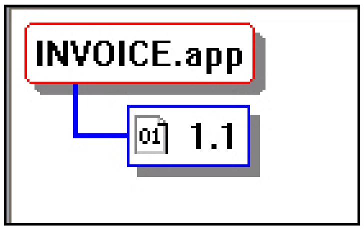


Figure 6. Invoice after initial import

Now, if you open the Modules file from your CVSROOT directory, you will see that it includes some kind of help text, but as with many Unix-originated documents, you can't say it's too user friendly.

I am interested in the -a option, aliases. Here is what my Invoice module looks like (line breaks added):

```
INVOICE -a Invoice Clarion6/BIN Clarion6/Template
  Clarion6/LIB | Clarion6/LIBSRC  Clarion6/images
  Clarion6/3rdParty/Template | Clarion6/3rdParty/bin
  Clarion6/3rdParty/libsrc
```

As you can see, it's a simple list of directories. Be careful – these *must* be the full directory paths from *inside* your CVS repository, *not* your machine! And if your server is Linux, you had better type the exact case, or you will have errors.

Let's say that after some last modifications to Invoice.app, I am ready for release. To mark the big event from a CVS point of view, I want to "tag" this module, which is like taking a snapshot of the state of each file in all those directories.

To do this, I go to the Remote menu and select the Create a tag by module option. To be sure that I select the INVOICE Module, not just my Invoice directory, I click on the lookup button on the right of the drop down list box, to access the server defined modules, and enter the name for my tag (Note that spaces and dots are not allowed).

Now, be careful! With CVS (I am not sure about CVSNT) there is tagging and then there is tagging! If you tag by Module, using the Remote menu option, you tag on the *server*, that is, the tag will be put on the HEAD (last revision), which seems pretty normal. *But sometimes that's not what you want!* Example: You carefully update Clarion 6.1 as soon as SoftVelocity release a hotfix, test it and develop with it, as I described above. But, you have another directory(or machine) for your production application, which you don't update to the latest, preferring to release from a stable version. In that case, you would use Modify/Create a tag, rather than Remote/Create a tag by module.

Modify will tag relative to the currently active version, indicated in the graph by the little document icon, rather than systematically to the HEAD (The HEAD being always the last version, active or not). This means that, if you are in a multi-developer environment, you might very well be polishing a new version, for which you are using Clarion 6.1, built 9028 (for example), while on the server, the last version is Clarion 6.1 build 9032. You might very well use this latest version from another machine, for tests and maybe development of a new application. For your current work, if you develop with 9028, you want to tag 9028 with your production files, which is what I mean by "the currently active version". If this is the case, you will want to use Modify/Create a tag, as Remote/Create a tag would tag 9032. It might sound more complex than it really is. Just sit in front of WinCVS on your machine, tests those options, and you will quickly understand what this is all about.

Release

Here is the graph for Invoice.App immediately after release:

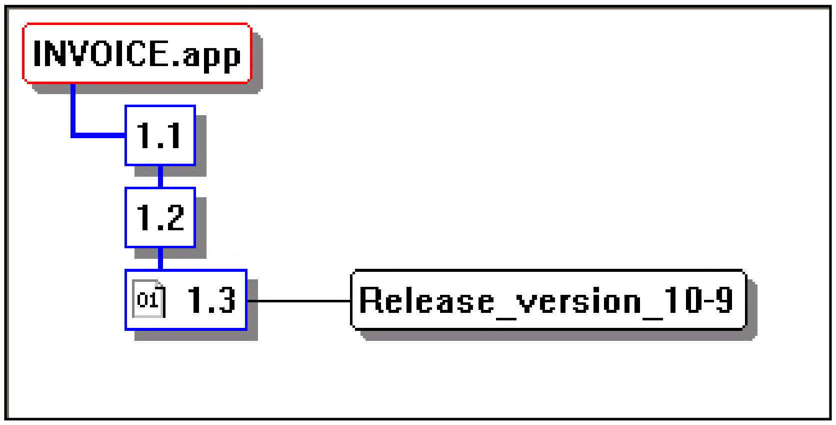


Figure 7. Invoice after tagging for release 10-9

Notice, on Figure 7, the little icon document with "01" inside. It shows that 1.3 is the currently active revision, and the file is binary. It's your clue, if you have complex graphs with branches and tagging, of where you are on the tree.

Now, I work a little on my next invoice version, and commit revisions 1.4 and 1.5.

Now that I have updated and committed my work on Invoice.app, the phone rings, my wealthiest (and best) client calls with a major showstopper bug found in Invoice 10.9. I need a fix a.s.a.p. Problem is, I have some new and un-tested classes in my new version that I introduced immediately after the release, as well as a few beta releases of third party tools. I can't afford to fix the bug and send something to that client with all those un-tested changes. Solution? Branching.

Branching Out

It's time to "Branch-Out", using my "release tag" as point from which to branch.

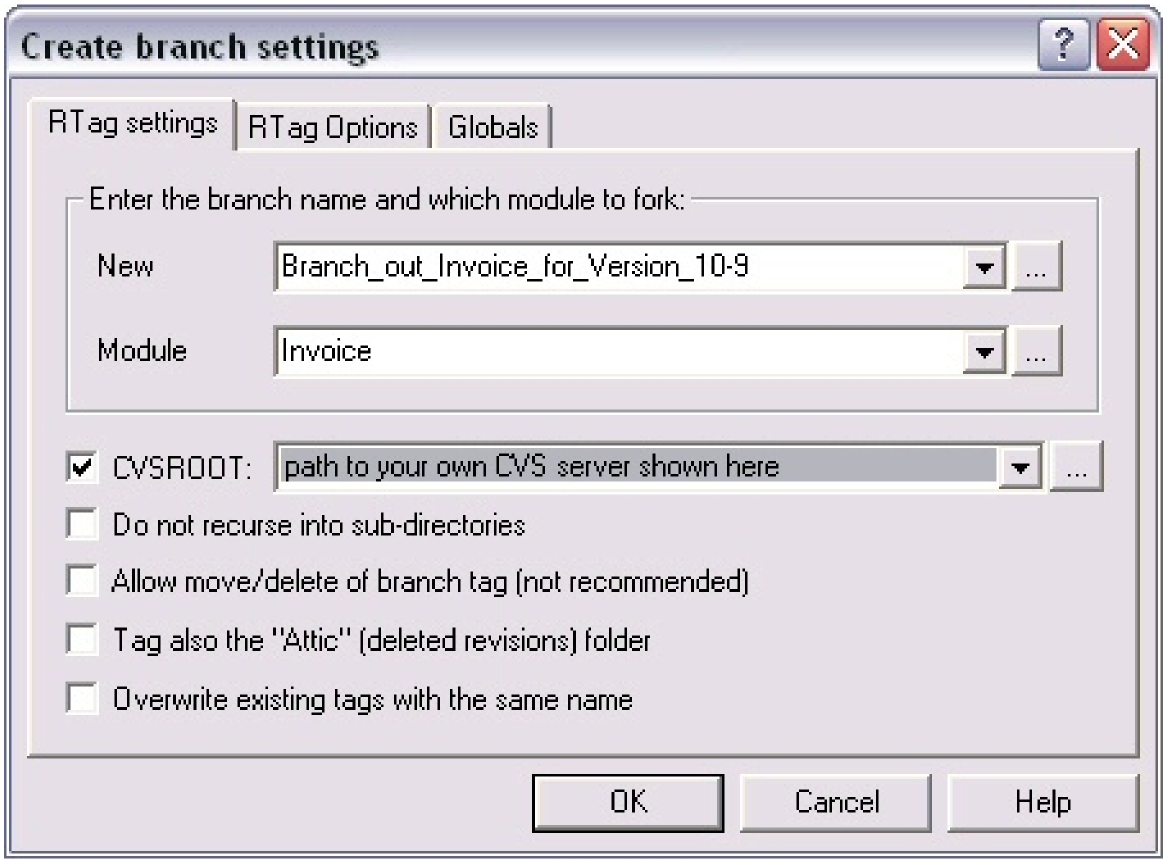
I need to rename my working directories before I make changes for this branch version, or else when I checkout code for the branch, I will overwrite my working files. In CVS terms, I need a new "sand-box"!

Practically, for the time where I will need to work on both my old version and the current version, I will have an "Invoice_Original" directory, as well as a "Clarion6_Original" directory. If I have defined my INVOICE module properly, checking it out should recreate everything I need to work with my version at the time, so no worries, and my paths and redirection file will stay correct. Now, you might wonder why it would be needed to create a new "Sandbox" when I branch out?

The reason is that CVS does not treat binary files and text files equally. When I checkout the branch, everything will be fine. But when I will want to go back to the trunk, that is my last revision, out of the branch, after committing my changes on the branch, if I do an "update", CVS will MERGE my text files, and not replace them, like it does for binaries, and that might very well NOT be what I want! It is much better to work with totally separated directories, and merge manually exactly what I decide to merge.

> **Important**: Do not forget to commit your redirection file and .C6ee.ini file, if you want to find your own settings when you will have to checkout an older branch of Clarion.

Now that I have renamed my directories, I go to CVS Remote menu, and select the Create a Branch option, as seen on Figure 8:



Enter the branch name and which module to fork:

New: Branch_out_Invoice_for_Version_10-9

Module: Invoice

[✓] CVSROOT: path to your own CVS server shown here

[ ] Do not recurse into sub-directories

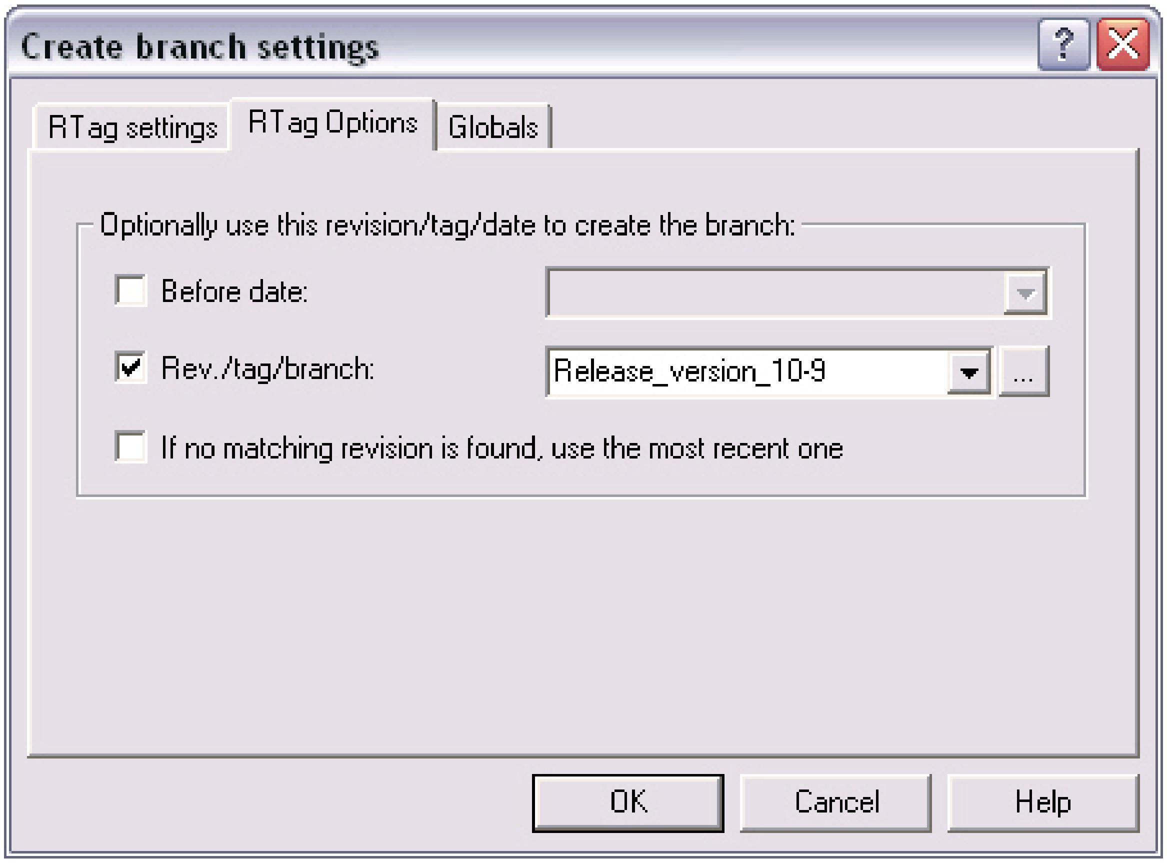
[ ] Allow move/delete of branch tag (not recommended)

[ ] Tag also the "Attic" (deleted revisions) folder

[ ] Overwrite existing tags with the same name

Figure 8. Create Branch settings, tab Rtag Settings

Most importantly, I use the second tab, as seen on Figure 9, to select the point from which I am going to branch out:



Optionally use this revision/tag/date to create the branch:

[ ] Before date:

[✓] Rev./tag/branch: Release_version_10-9

[ ] If no matching revision is found, use the most recent one

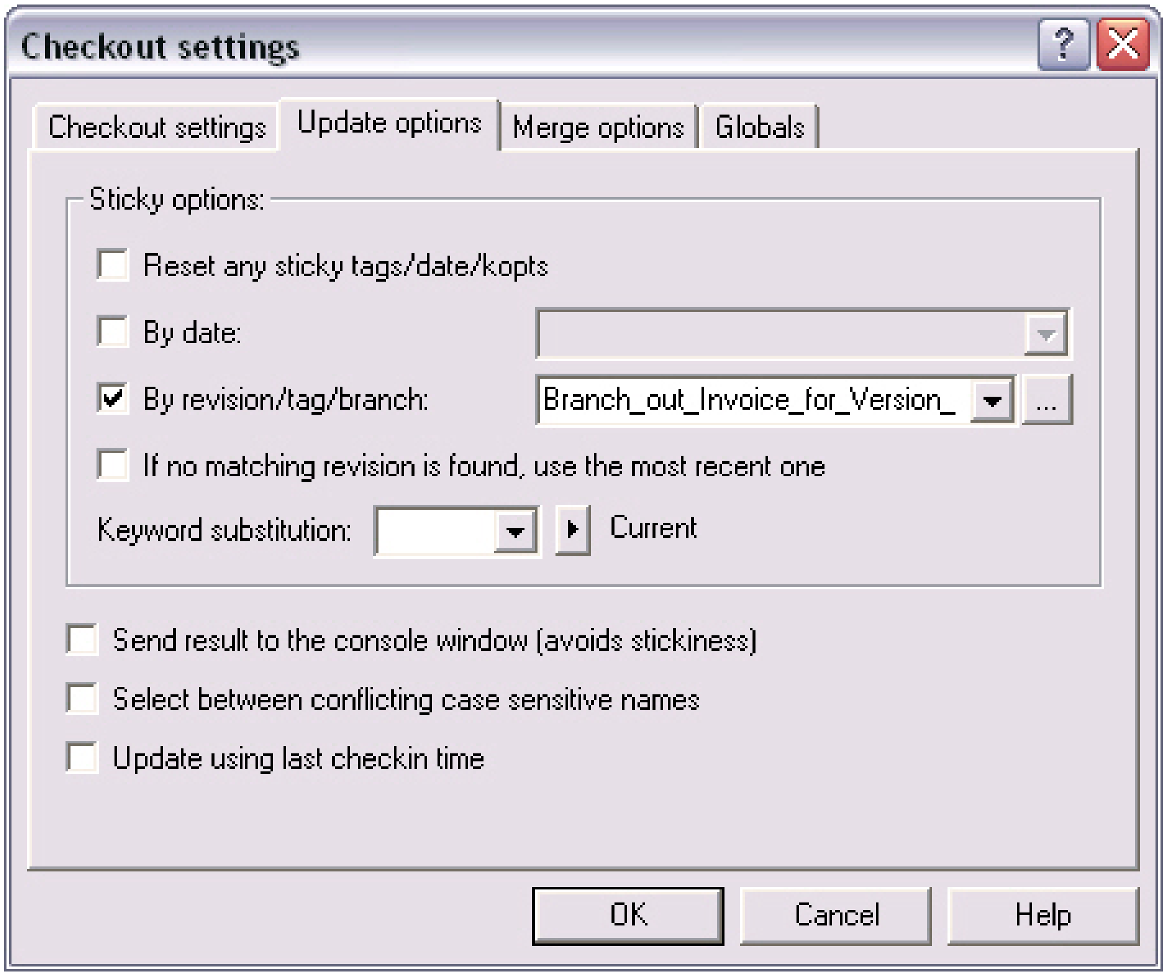
Figure 9. Create Branch settings, Tab 2, tag Options

CVS lets me know that the branching went well (line breaks added).

```
cvs -d :pserver;username=xxxx;password=xxxx;hostname=
  192.168.1.105:/home/cvsroot... rtag -b -r Release_version_10-9
  Branch_of_Invoice_Version_10-9 INVOICE
cvs rtag: Tagging Invoice
cvs rtag: Tagging Invoice/HELPSRC
cvs rtag: Tagging Clarion6/BIN
cvs rtag: Tagging Clarion6/BIN/Flash
cvs rtag: Tagging Clarion6/Template
cvs rtag: Tagging Clarion6/3rdParty/Template/_BACKUP_
cvs rtag: Tagging Clarion6/3rdParty/bin
cvs rtag: Tagging Clarion6/3rdParty/libsrc
***** CVS exited normally with code 0 *****
```

It might surprises you to see this working just fine, without any error, when I just renamed my directories! The branch/tag happened on the server, in the repository – the branch files don't yet exist on the local drive.

To work on this older version, I need to checkout this branch. I go to the Remote menu, checkout module (INVOICE), and on the second tab "Update options", as seen on Figure 10, , I select my branch:



Sticky options:

- [ ] Reset any sticky tags/date/kopts
- [ ] By date:
- [✓] By revision/tag/branch: Branch_out_Invoice_for_Version_
- [ ] If no matching revision is found, use the most recent one

Keyword substitution: Current

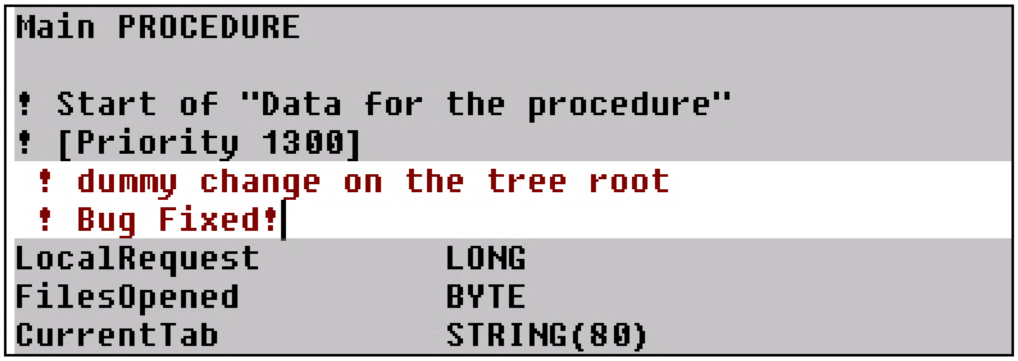
- [ ] Send result to the console window (avoids stickiness)
- [ ] Select between conflicting case sensitive names
- [ ] Update using last checkin time

Figure 10. Checkout by branch.

Once you click OK, CVS re-creates the directories on your machine, and you are back in the environment you had at the time you tagged your release!

Bug FIX!

For demonstration purposes, my bug fix is actually just a comment:



```
Main PROCEDURE

! Start of "Data for the procedure"
! [Priority 1300]
 ! dummy change on the tree root
 ! Bug Fixed!
LocalRequest           LONG
FilesOpened            BYTE
CurrentTab             STRING(80)
```

Figure 11. Bug Fixed!

Fast and simple…but an excellent way to see what CVS does, even if this is a binary file, although CVS is much better dealing with pure text files (you will see later what is at the same place in my file AFTER release).

Let's have a look at the graph, too:

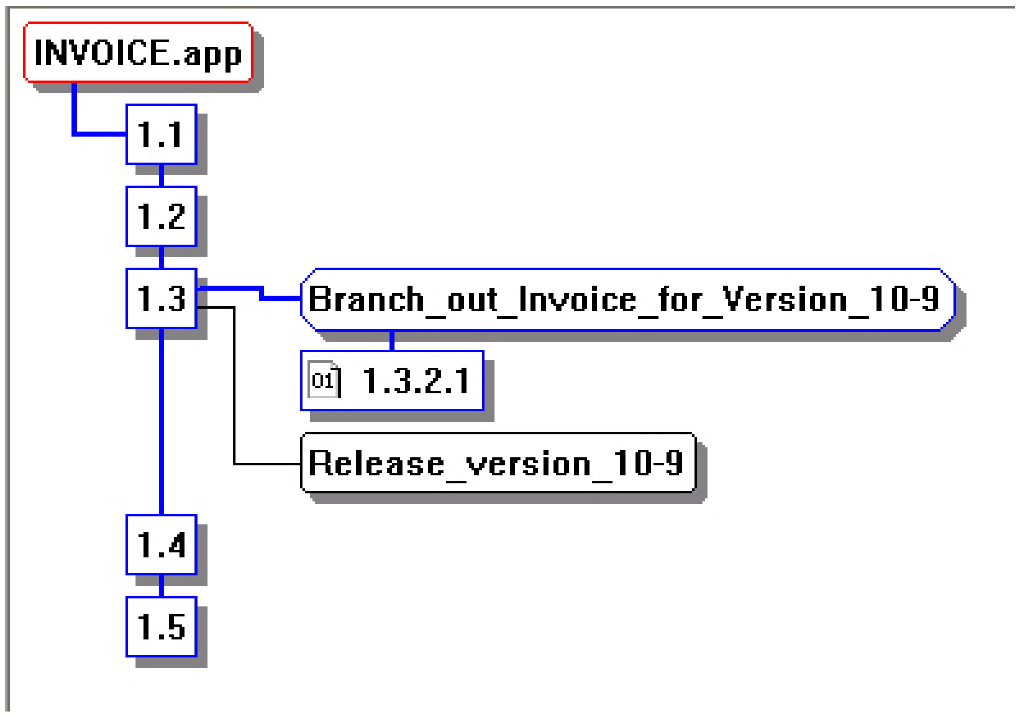


Figure 12. Bug Fixed on invoice branch.

During the commit, CVS lets me know I am actually working on the branch, telling me that the new revision is 1.3.2.1, while the previous version was 1.3. the extra digits tells me that I am not on the trunk anymore, and you might see on the graph the little document icon indicating the active revision is on 1.3.2.1, which is the first revision of the branch

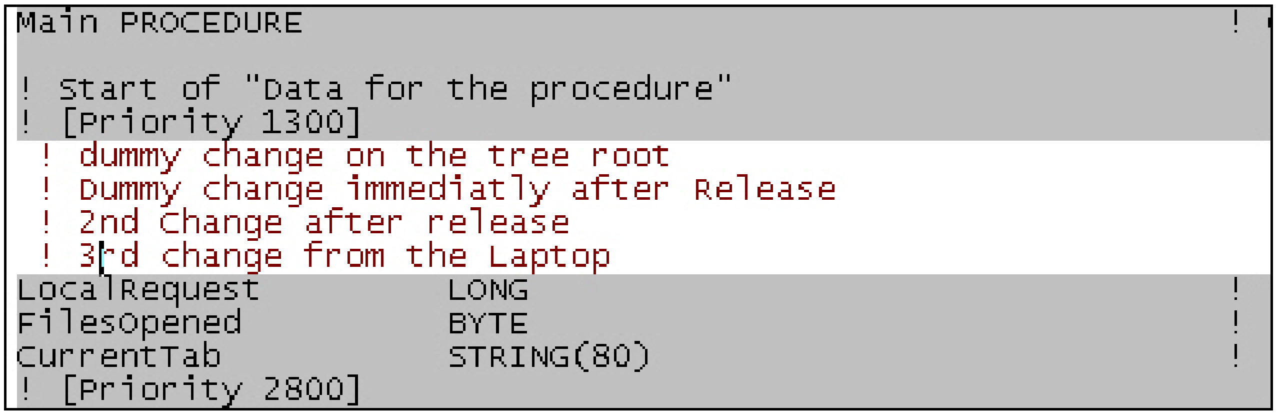
```
    cvs commit -m "Bug Fixed!" invoice.app (in directory D:\Invoice\)
```

Checking in invoice.app;

```
    /home/cvsroot/repository/Invoice/invoice.app,v <-- invoice.app
    new revision: 1.3.2.1; previous revision: 1.3
    done
    ***** CVS exited normally with code 0 *****
```

Now what happen if another developer on another machine continues working on the new version, not knowing I am working on the previous version? Nothing changes, as his active branch is still the main trunk, and its commit would create revision 1.6.

Once you are done fixing your bug(s), compiling and delivering the fix to your client(s) you check your changes in to CVS. Then you can delete your Invoice and Clarion working directories, rename Invoice_original and Clarion6_original back to Invoice and Clarion, update your module, just to be safe, and carry on….. Let's say I made some changes on the version I have installed on my laptop, as seen on Figure 13. Here's the code I committed:



```
Main PROCEDURE                                                    !

! Start of "Data for the procedure"
! [Priority 1300]
 ! dummy change on the tree root
 ! Dummy change immediatly after Release
 ! 2nd Change after release
 ! 3rd change from the Laptop
LocalRequest              LONG                                    !
FilesOpened               BYTE                                    !
CurrentTab                STRING(80)                              !
! [Priority 2800]
```

Figure 13. Last change made from the Laptop

No bug fix here – this is a new feature, so this branch is really separated from the main trunk.

And the matching graph on Figure 14 after I update from my main machine:

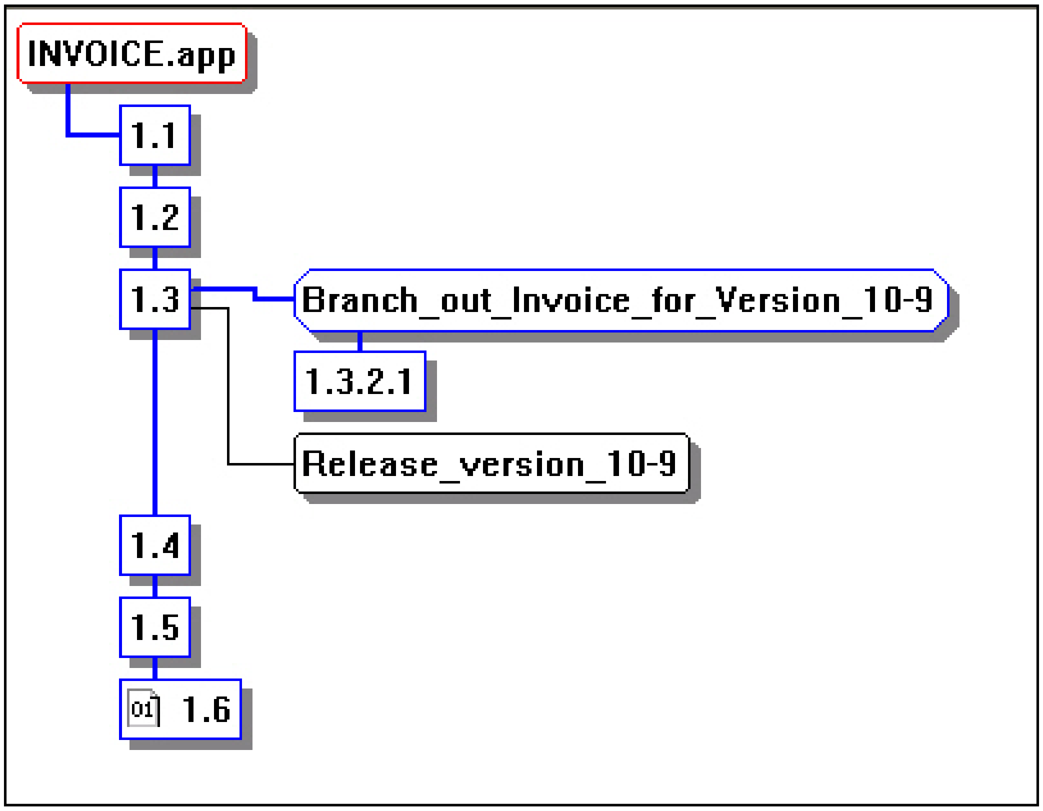


Figure 14. Invoice after change 6 on the Trunk.

But, you are going to ask, what if I want to MERGE the fix that I did for the old version with the new one?

Well, CVS allows that also. It's called "pruning back" to the main tree. There are some cases where it might be needed and convenient to do so, but I would *never* attempt this at a whole module level, as CVS doesn't know how to merge binaries. If you were actually trying this with APP files and or DLLs/EXEs, CVS would rename your current files, and bring back the branch version! Not good at all!

If you want to merge individual text files, you can update, with the option to "reset any sticky tags", and "merge with one rev/tag". This creates a new revision as a result of the merge that you will need to commit.

Now that you have a pretty decent idea of what's possible to do with CVS, I hope you are impatient to install it and start using it, if you have not already done so. Next week I'll describe how to set up a server-based CVS installation; if you want to install CVS locally, refer to Nardus's article.

Bernard Grosperrin is a native of France, and has been a big fan of everything American since his teens. He began visiting the US in 1996, and moved to San Antonio in 1998, where he discovered his love of Mexican food. He and his lovely wife Gloria, who he met in Tulsa, now live in California. Bernard has been programming and designing software and databases for 14 years, primarily with Clarion. His hobbies include flying radio-controlled airplanes and riding his motorcycle. He loves aircraft of all kinds, and can't miss an opportunity to fly, whether it's in a glider, a World War II trainer, or a general aviation Piper or Cessna.

Reader Comments

Add a comment

Clarion Magazine

Version Control with CVS and Clarion 6.x, Part 2

by Bernard Grosperrin

Published 2005-04-21

Last week I explained the concepts behind CVS, and how these apply to Clarion developers. This week I'll cover the CVS server and client installation and configuration in more detail.

I am not going to fully detail an installation for the server, as there are too many specificities, depending on your own local configuration, needs, operating system, company policies, etc. I will just give you some pointers as well as some directions you can take to improve on a basic install. For example, I will not expand much on security, but I would suggest, mostly if you want to have collaborative work through the Internet, to look at SSH tunneling.

My own server runs on Fedora Core 3 from Red Hat, so you may have to adapt these instructions for other Linux distributions.

For Windows, installing is pretty straightforward, but I would suggest reading the documentation anyway if you want things to run smoothly.

I recommend you do not use a shared directory for your repository. You need a *server*, and it would be a lot better and safer in the long run if your repository was not visible in your network neighborhood. WinCVS 1.3 will not accept to work on a shared directory anyway.

You probably have an old machine somewhere you don't know how to recycle, or a computer you let the kids play with. This could make a pretty decent CVS server with Linux, if you are diplomatic enough...

I just switched from using CVS (which is part of Fedora Core 3 distribution), to CVSNT, so this article is all about CVSNT, and I will try to point out differences when necessary. (I recommend using CVSNT over CVS)

CVSNT Install on Red Hat Fedora Core 3

As CVS is part of the Red Hat Fedora Core 3 distribution, I will simply upgrade the already-installed CVS to CVSNT. There is a perfect upward compatibility between CVS and CVSNT, so it's not a big deal to switch, even if you are already using CVS extensively. CVSNT Wiki has a pretty good explanation, which I followed step by step: http://www.cvsnt.org/wiki/InstallationLinux

First, don't forget to remove your pre-existing CVS installation with the command:

rpm –e cvs

If you are installing CVSNT on a machine that has no previous CVS installation, you should at first create a directory to be your CVS repository, as well as a cvs user and group, and give the cvs group all the rights to the cvs directory tree.

You can install a binary RPM, but I prefer to install sources when available, and build the application on my machine.

Then from inside your cvsntxxxx directory, build CVSNT from the source with the following three commands:

```
./configure
make
make install
```

Depending on your hardware this can take a bit of time, but it really works just fine, and ensures that CVSNT is correctly built for your particular Linux installation.

As indicated by the CVSNT Wiki page, rename and modify the Pserver file. You just have to indicate the directory that you are going to use as CVS repository.

Then, you have to build up your cvspserver file. Here is mine (line breaks added):

```
service cvspserver
{
    disable    = no
    socket_type = stream
    protocol = tcp
    wait = no
    user = root
      passenv = PATH
      server = /usr/local/bin/cvs
    server_args = -f --allow-root=/home/cvsroot/repository
        --allow-root=/home/cvsroot/public_repos pserver
}
```

Obviously, you will have to adapt to your own paths. Restart xinetd (using the command service xinetd restart), and you should have access to your new CVSNT server!

Continue to follow the CVSNT Wiki page for Lockserver and the repository setup if this is a new install for you. *Don't init your repository if it already exists !*

If you need/want to install SSH, you need to do so now. If you are going to work from your own local network, without external access, you don't need to worry about SSH.

Now it's time to set up the WinCVS client.

WinCVS install and configuration.

Nardus explained the WinCVS installation in his article, and I encourage you to read it now if you haven't already. Just a little remark: his install is on a standalone windows machine. You don't need a full CVSNT install if you have a separate server, and you will not have the Locking service running on your machine, as it has to be running on the server.

You can set a `CVSROOT` environment variable, but I don't really recommend doing so, as WinCVS allow you to work with multiple servers; as you can very well have your own in-house server for your own work, an outside server for some work you are collaborating on, and maybe a few open-source CVS servers. (For instance, you can checkout the sources for WinCVS and compile it yourself if you have Visual C++ 6.0). All the settings are kept on a per directory basis. WinCVS will ask you the settings when they are needed, but most of the commands use what is stored in your (hidden) CVS directory created each time you checkout a module.

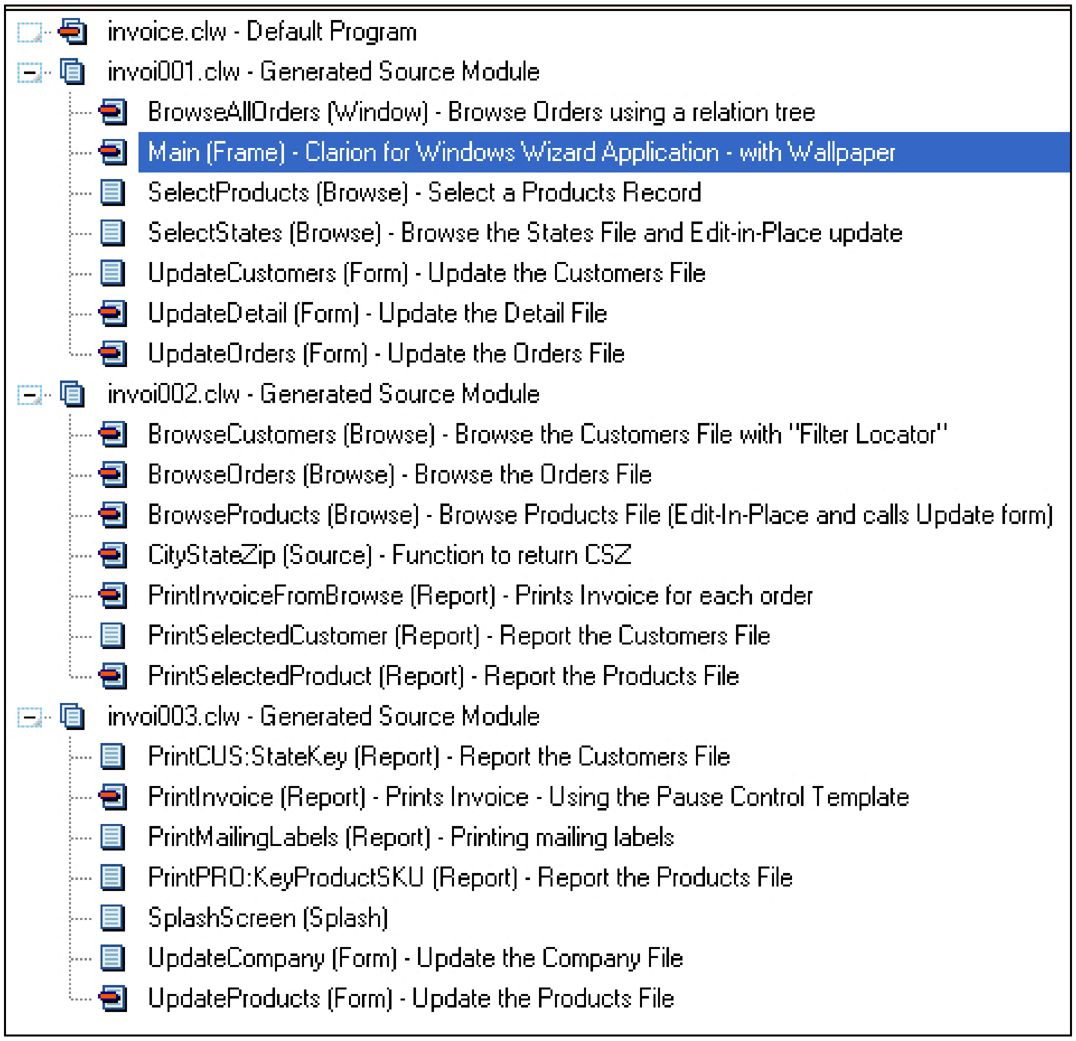
Configure your Applications and Directories to work with Clarion6 and CVS

You know that Clarion, unlike many other tools, does not store all source in ASCII files (unless you program entirely by hand rather than using APPs and DCTs). And CVS doesn't know how to merge binary files that well. Luckily enough, with Clarion 6 SoftVelocity introduced a way to customize the Version Control feature, so that it can work with CVS, among other versioning software.

When you use the Version Control interface, Clarion will create a TXA file (renamed APV) for each module in your application, plus one for Applications Options, as well as a TXD File (renamed DCV) for your dictionary.

I suggest that you rename each of your modules in a more meaningful way than Appli001, Appli002, etc, if you want to know what each file is and what it does! I will demonstrate this with the Clarion Invoice application that I previously put under Version control the "brute force" way, that is the binary app file itself.

As seen on Figure 15, Invoice has only four modules, Invoice.clw, Inv001.clw, Inv002.clw and Inv003.clw.



```
invoice.clw - Default Program
invoi001.clw - Generated Source Module
    BrowseAllOrders (Window) - Browse Orders using a relation tree
    Main (Frame) - Clarion for Windows Wizard Application - with Wallpaper
    SelectProducts (Browse) - Select a Products Record
    SelectStates (Browse) - Browse the States File and Edit-in-Place update
    UpdateCustomers (Form) - Update the Customers File
    UpdateDetail (Form) - Update the Detail File
    UpdateOrders (Form) - Update the Orders File
invoi002.clw - Generated Source Module
    BrowseCustomers (Browse) - Browse the Customers File with "Filter Locator"
    BrowseOrders (Browse) - Browse the Orders File
    BrowseProducts (Browse) - Browse Products File (Edit-In-Place and calls Update form)
    CityStateZip (Source) - Function to return CSZ
    PrintInvoiceFromBrowse (Report) - Prints Invoice for each order
    PrintSelectedCustomer (Report) - Report the Customers File
    PrintSelectedProduct (Report) - Report the Products File
invoi003.clw - Generated Source Module
    PrintCUS:StateKey (Report) - Report the Customers File
    PrintInvoice (Report) - Prints Invoice - Using the Pause Control Template
    PrintMailingLabels (Report) - Printing mailing labels
    PrintPRO:KeyProductSKU (Report) - Report the Products File
    SplashScreen (Splash)
    UpdateCompany (Form) - Update the Company File
    UpdateProducts (Form) - Update the Products File
```

Figure 15. Invoice Modules "out of the box"

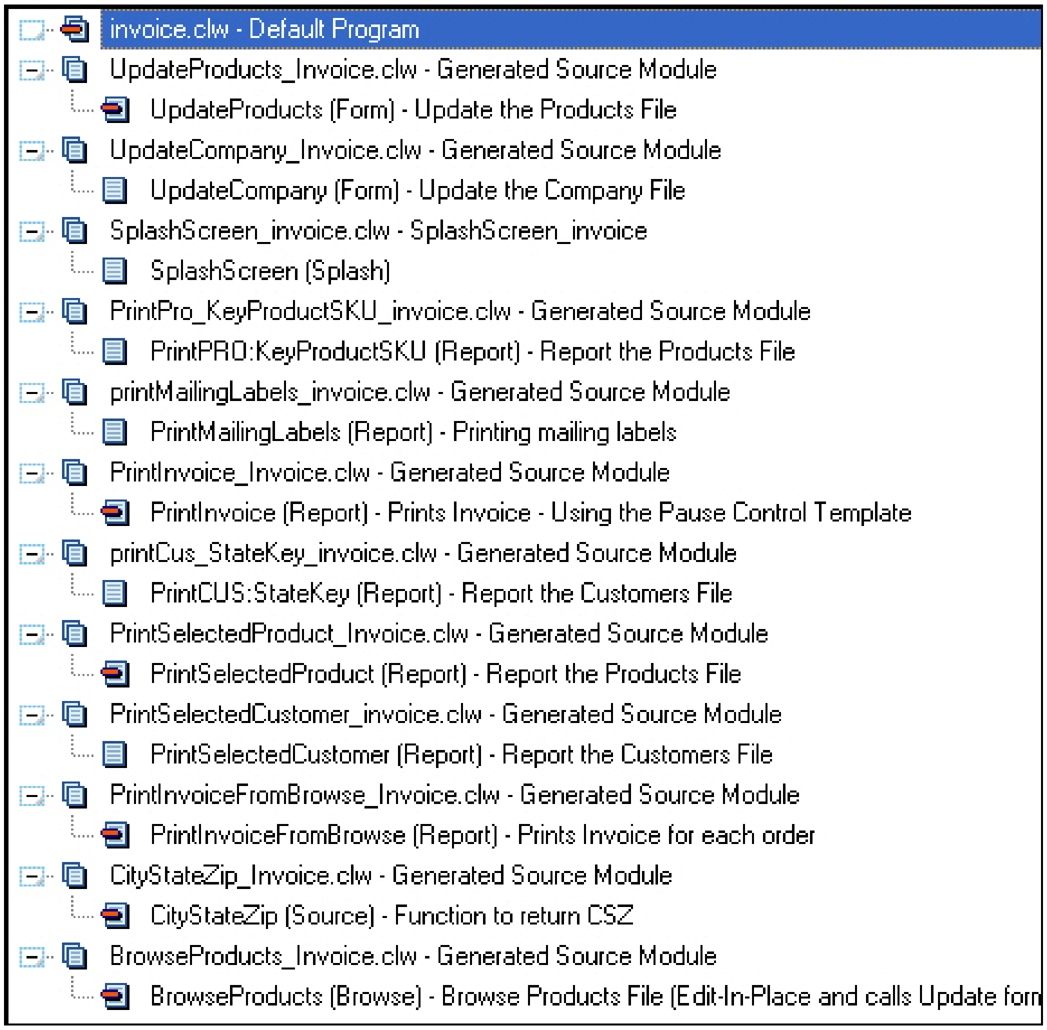
Each module has about seven procedures. This will not help me much if I have three or four programmers working together, as they all would have to commit the same module once they are done. And the file names do not make it clear where `PrintInvoice` is, for example.

So the very first thing to change, in order to make it easier and more efficient to use CVS with Clarion, is to split my application in more modules, and give them meaningful names. I don't absolutely need to have *one* procedure per module, but it's better if I can reach that level of granularity. (I am not trying to say that it is mandatory to rename all your modules, version control will work just fine without doing this, but, in the long run, you will thank me for this suggestion.)

To manually change your modules in a Clarion application, select the procedure you want to change, go to the procedure menu, select change module, then select New Module, give the name you want, and validate. It's as simple as that!

Okay, I know it's a long and tedious job, mostly if you have hundreds of procedures per application in 30 or 50 apps! But it's time well spent, and it will pay back very quickly.

You can see in Figure 16 below how the application looks like after this transformation.

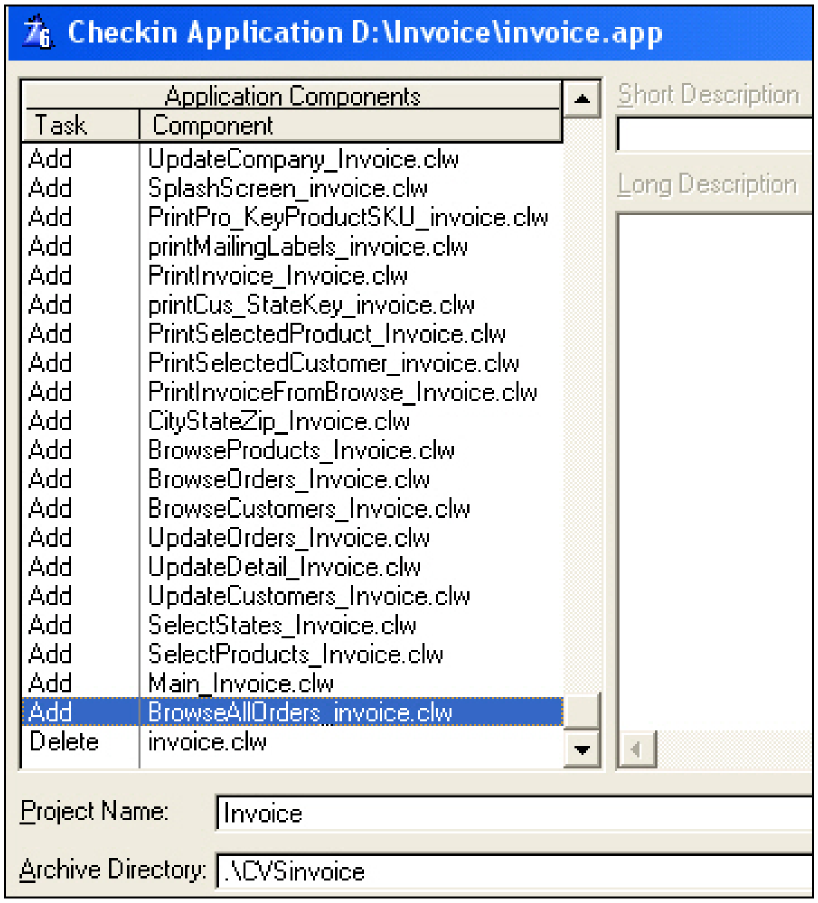


```
invoice.clw - Default Program
UpdateProducts_Invoice.clw - Generated Source Module
    UpdateProducts (Form) - Update the Products File
UpdateCompany_Invoice.clw - Generated Source Module
    UpdateCompany (Form) - Update the Company File
SplashScreen_invoice.clw - SplashScreen_invoice
    SplashScreen (Splash)
PrintPro_KeyProductSKU_invoice.clw - Generated Source Module
    PrintPRO:KeyProductSKU (Report) - Report the Products File
printMailingLabels_invoice.clw - Generated Source Module
    PrintMailingLabels (Report) - Printing mailing labels
PrintInvoice_Invoice.clw - Generated Source Module
    PrintInvoice (Report) - Prints Invoice - Using the Pause Control Template
printCus_StateKey_invoice.clw - Generated Source Module
    PrintCUS:StateKey (Report) - Report the Customers File
PrintSelectedProduct_Invoice.clw - Generated Source Module
    PrintSelectedProduct (Report) - Report the Products File
PrintSelectedCustomer_invoice.clw - Generated Source Module
    PrintSelectedCustomer (Report) - Report the Customers File
PrintInvoiceFromBrowse_Invoice.clw - Generated Source Module
    PrintInvoiceFromBrowse (Report) - Prints Invoice for each order
CityStateZip_Invoice.clw - Generated Source Module
    CityStateZip (Source) - Function to return CSZ
BrowseProducts_Invoice.clw - Generated Source Module
    BrowseProducts (Browse) - Browse Products File (Edit-In-Place and calls Update forn
```

Figure 16. Invoice Modules after transformation for CVS.

Now I can look at my files and have a pretty good idea of what procedure they contain, and what they are supposed to do.

On your file menu in Clarion, you may have noticed the new options Check In and Check Out. Click on Check In, as seen on Figure 17.


Checkin Application D:\Invoice\invoice.app

Task	Component
Add	UpdateCompany_Invoice.clw
Add	SplashScreen_invoice.clw
Add	PrintPro_KeyProductSKU_invoice.clw
Add	printMailingLabels_invoice.clw
Add	PrintInvoice_Invoice.clw
Add	printCus_StateKey_invoice.clw
Add	PrintSelectedProduct_Invoice.clw
Add	PrintSelectedCustomer_invoice.clw
Add	PrintInvoiceFromBrowse_Invoice.clw
Add	CityStateZip_Invoice.clw
Add	BrowseProducts_Invoice.clw
Add	BrowseOrders_Invoice.clw
Add	BrowseCustomers_Invoice.clw
Add	UpdateOrders_Invoice.clw
Add	UpdateDetail_Invoice.clw
Add	UpdateCustomers_Invoice.clw
Add	SelectStates_Invoice.clw
Add	SelectProducts_Invoice.clw
Add	Main_Invoice.clw
Add	BrowseAllOrders_invoice.clw
Delete	invoice.clw

Project Name: Invoice

Archive Directory: .\CVSinvoice

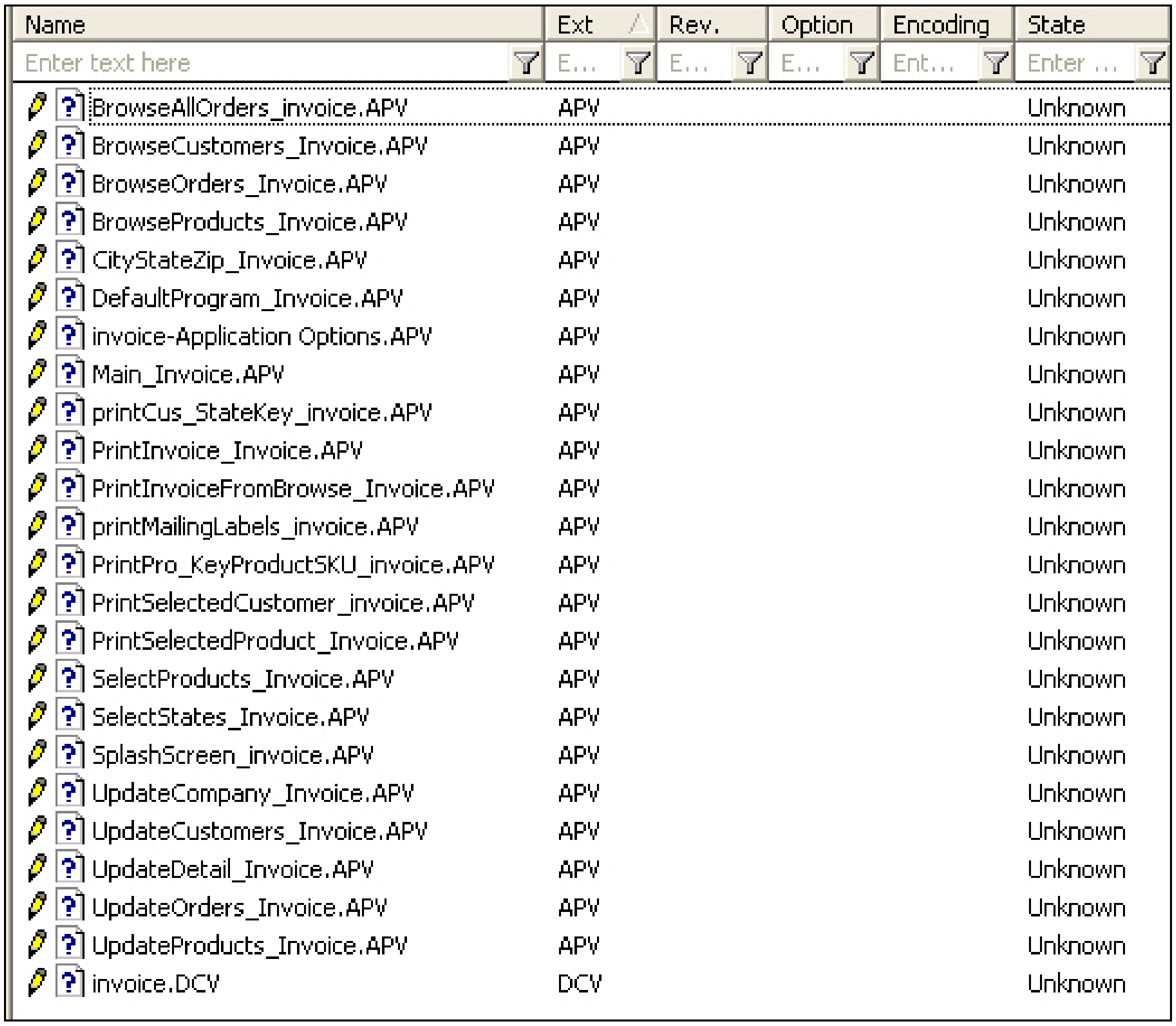
Figure17. Checkin Dialog from Clarion.

Figure 17 shows all my modules with the task "add" in front of them. This seems pretty logical, as I am just creating these modules and they are not yet under version control. The only thing I have done so far to configure Clarion with CVS is to select CVS (Button Configure CVS, then load commands, select CVS), and put a "rem" on the line in front of each set of commands! As these commands are all command line, a `rem` instruction will just disable any command, but Clarion will still create my APV and DPV files, and for now that's *all* what I want from Clarion. I'll do the rest in CVS.

Notice that I use a subdirectory named CVSApplicationName as the Archive directory for Clarion, so that the APV and DCV files are not mixed with my other files in my working directory.

So, what happens now if I click OK on this dialog? I will see the DOS command window pop up a few times, and not much more. Once finished, I can see in my CVSInvoice\Invoice subdirectory that I have my APV files, one per module. I still need to do the same for my dictionary, to get a DCV file the same way.

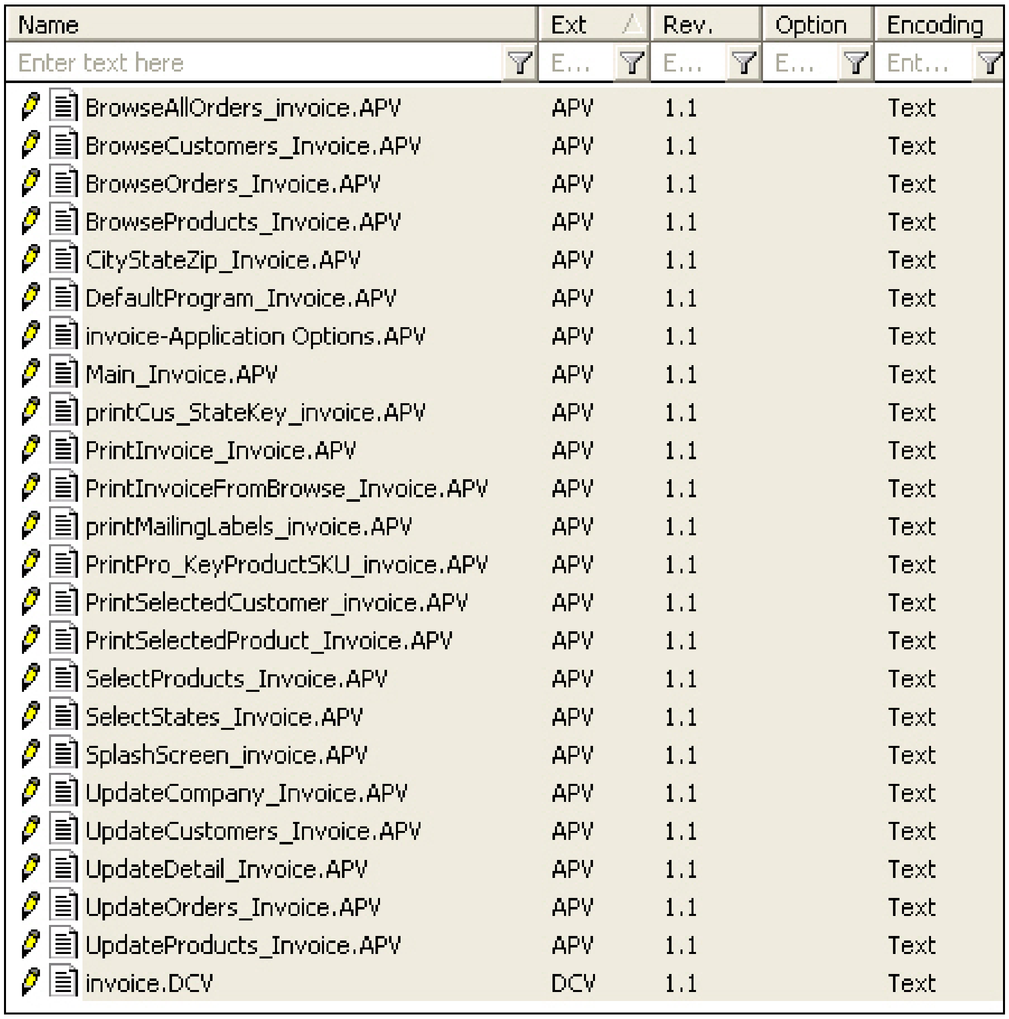
As this new sub-directory is not under version control, yet, I need to import it. Clarion will (if I wish) allow me to commit changes, but I first need to repeat the same process explained by Nardus in his First Import paragraph: Select my CVSInvoice Sub-directory, right click, select import, OK the filter, check the box to not create vendor tag, and to create CVS directories while importing.



Name	Ext	Rev.	Option	Encoding	State
Enter text here	E...	E...	E...	Ent...	Enter ...
BrowseAllOrders_invoice.APV	APV				Unknown
BrowseCustomers_Invoice.APV	APV				Unknown
BrowseOrders_Invoice.APV	APV				Unknown
BrowseProducts_Invoice.APV	APV				Unknown
CityStateZip_Invoice.APV	APV				Unknown
DefaultProgram_Invoice.APV	APV				Unknown
invoice-Application Options.APV	APV				Unknown
Main_Invoice.APV	APV				Unknown
printCus_StateKey_invoice.APV	APV				Unknown
PrintInvoice_Invoice.APV	APV				Unknown
PrintInvoiceFromBrowse_Invoice.APV	APV				Unknown
printMailingLabels_invoice.APV	APV				Unknown
PrintPro_KeyProductSKU_invoice.APV	APV				Unknown
PrintSelectedCustomer_invoice.APV	APV				Unknown
PrintSelectedProduct_Invoice.APV	APV				Unknown
SelectProducts_Invoice.APV	APV				Unknown
SelectStates_Invoice.APV	APV				Unknown
SplashScreen_invoice.APV	APV				Unknown
UpdateCompany_Invoice.APV	APV				Unknown
UpdateCustomers_Invoice.APV	APV				Unknown
UpdateDetail_Invoice.APV	APV				Unknown
UpdateOrders_Invoice.APV	APV				Unknown
UpdateProducts_Invoice.APV	APV				Unknown
invoice.DCV	DCV				Unknown

Figure 18. cvsinvoice\Invoice before adding the files to CVS, all flagged as unknown.

Once done, CVS shows me the directory with my files now having a revision number, as seen on Figure 19:



Name	Ext	Rev.	Option	Encoding
Enter text here	E...	E...	E...	Ent...
BrowseAllOrders_invoice.APV	APV	1.1		Text
BrowseCustomers_Invoice.APV	APV	1.1		Text
BrowseOrders_Invoice.APV	APV	1.1		Text
BrowseProducts_Invoice.APV	APV	1.1		Text
CityStateZip_Invoice.APV	APV	1.1		Text
DefaultProgram_Invoice.APV	APV	1.1		Text
invoice-Application Options.APV	APV	1.1		Text
Main_Invoice.APV	APV	1.1		Text
printCus_StateKey_invoice.APV	APV	1.1		Text
PrintInvoice_Invoice.APV	APV	1.1		Text
PrintInvoiceFromBrowse_Invoice.APV	APV	1.1		Text
printMailingLabels_invoice.APV	APV	1.1		Text
PrintPro_KeyProductSKU_invoice.APV	APV	1.1		Text
PrintSelectedCustomer_invoice.APV	APV	1.1		Text
PrintSelectedProduct_Invoice.APV	APV	1.1		Text
SelectProducts_Invoice.APV	APV	1.1		Text
SelectStates_Invoice.APV	APV	1.1		Text
SplashScreen_invoice.APV	APV	1.1		Text
UpdateCompany_Invoice.APV	APV	1.1		Text
UpdateCustomers_Invoice.APV	APV	1.1		Text
UpdateDetail_Invoice.APV	APV	1.1		Text
UpdateOrders_Invoice.APV	APV	1.1		Text
UpdateProducts_Invoice.APV	APV	1.1		Text
invoice.DCV	DCV	1.1		Text

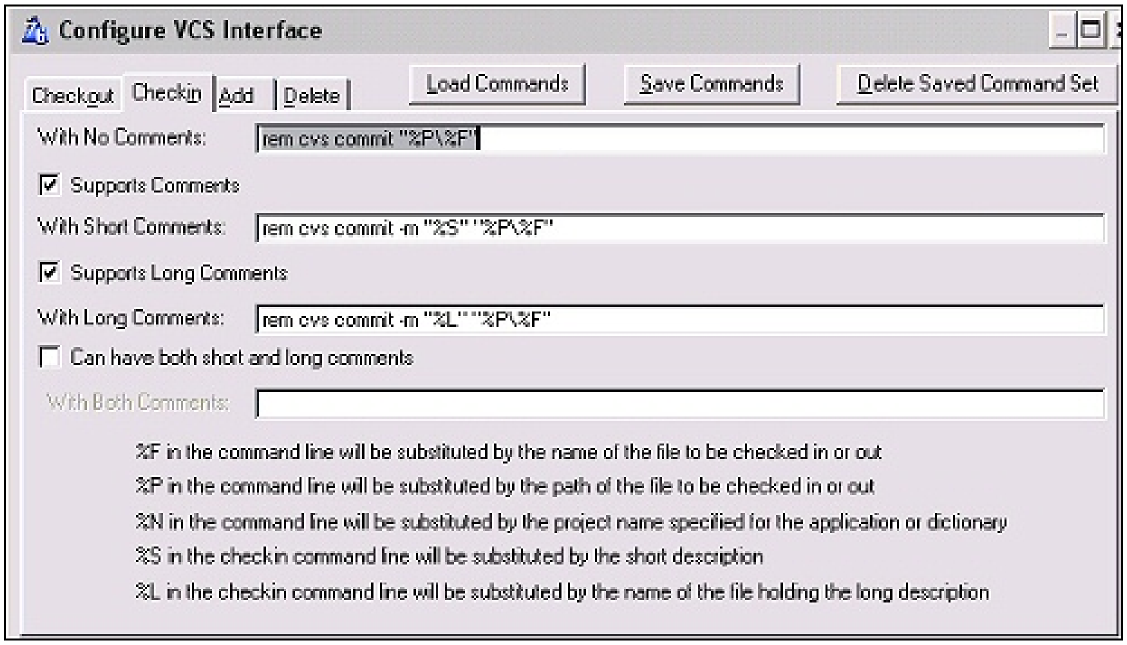
Figure 19. cvsInvoice\Invoice after adding files to CVS

After this initial import, I can, if I want to, configure Clarion so that I would be able to commit directly from Clarion. Everything has advantages and inconveniences, but I am sure many of you will want to be able to directly commit from Clarion. I will show to you how to do this, but I also like the control WinCVS gives me, and the capability to make a lot of operations automatic. At the end, you can judge which method seems better for you.

To make Clarion work properly with CVS, the easiest way is to look at the commands generated by WinCVS: For example, If I select commit for a file, WinCVS show this command in the console:

```
cvs commit -m "no message" UpdateCompany_Invoice.APV (in directory
D:\Invoice\CVSInvoice\)
```

So, in Clarion, I enter the exact same command, adapting to what Clarion indicates for comments, files and directories, as seen on Figure 20:


Configure VCS Interface

Checkout | Checkin | Add | Delete | Load Commands | Save Commands | Delete Saved Command Set

With No Comments: rem cvs commit "%P\%F"

[✓] Supports Comments

With Short Comments: rem cvs commit -m "%S" "%P\%F"

[✓] Supports Long Comments

With Long Comments: rem cvs commit -m "%L" "%P\%F"

[ ] Can have both short and long comments

With Both Comments:

%F in the command line will be substituted by the name of the file to be checked in or out

%P in the command line will be substituted by the path of the file to be checked in or out

%N in the command line will be substituted by the project name specified for the application or dictionary

%S in the checkin command line will be substituted by the short description

%L in the checkin command line will be substituted by the name of the file holding the long description

Figure 20. Commit settings in Clarion (view full sized image)

As indicated earlier, as I am making some experiments and tests, right now, I simply put `rem` in front of each command line to *not* commit directly,as in Figure 20. You will need to have the directory where CVS is located in your path, for Clarion to be able to use it. If you install WincvS 1.3 xx, this directory should be

```
    C:\Program Files\GNU\WinCvs 1.3\CVSNT
```

As you might have guessed, `Commit` in CVS terminology, is the same as `check in` for many other Version Control Systems. (The terminology is important. You don't "check in" with CVS, as you are free to continue working on your files after committing).

For "Add" the WinCVS command is add, as seen here (line break added):

```
cvs add BG_FILE_2.CLW BG_FILE_2.INC
  (in directory D:\Clarion6\LIBSRC\)
```

There are no comments for add with CVS, as the files are simply filtered and flagged to be committed. The comment will be included when you commit the files, so after adding your modules from Clarion, you need to go to WinCVS and commit your files from there.

Remove is like add, but with different consequences. Here is what the console displays on a delete (line breaks added):

```
'BrowseAllOrders_invoice.clw' has been
  moved successfully to the recycle bin...
cvs remove BrowseAllOrders_invoice.clw
  (in directory D:\Invoice\)
cvs server: scheduling
  `BrowseAllOrders_invoice.clw' for removal
cvs server: use 'cvs commit' to remove
  this file permanently
```

This command `remove` does two things: it deletes the file from your disk (in the Windows way, moves the file to the recycle bin), *and* it flags it for removal from the server repository. It's important to do both, as if you simply delete the file from your disk, the next time you update this directory, CVS will re-import it, and you might have some surprises when compiling if you end up with modules supposedly eliminated. (That is, unless you use the macros included in the source for this article – those macros read the project in the Application Options.APV file, avoiding any errors of this kind).

As with the `add` command, the files to be deleted have to be committed to be effectively removed from the repository.

The last set of commands to put into Clarion is for *checking-out*, or *updating* in CVS language.

```
cvs update -P Main_Invoice.APV
  (in directory D:\Invoice\CVSInvoice\)
```

Again, I will translate this command into Clarion with `cvs update -P "%P\%F"`

> **NOTE:** I never check out directly from Clarion, mostly because it takes too long, as I have to go to each module individually and set the option one for each one. Also, for some Applications, it seems that Clarion trashes the app file if I don't import the whole TXA at once, as when the "application Options" files is merged with the current one, the AppGen does not seem to find the embeds properly, and duplicates the code in orphan embeds.. Happily enough, the set of macros and Clarion utility I've included with this article makes updating your application(s) a breeze.

Directory Structure

As you might have guessed from the above, I work with a standard directory structure, to make it easier to not mix modules in a multi application product, and to make committing and updating as automatic as I possibly can. This structure is *mandatory* for my macros to run properly.

My directory tree looks like this:

```
Product
    Cvsproduct
        Appname
        Appname
```

All that is fine and dandy, but you might be a bit reluctant to adopt this same structure if you have a few products, and each one of them has 30-50 applications. That's a lot of work just to be able to put your applications under version control! WinCVS can be adapted more precisely to your tools and environment via macros written using Python, as seen on Figure 21, so I wrote a macro just for that, it takes one click and a few seconds to create the structure described above:

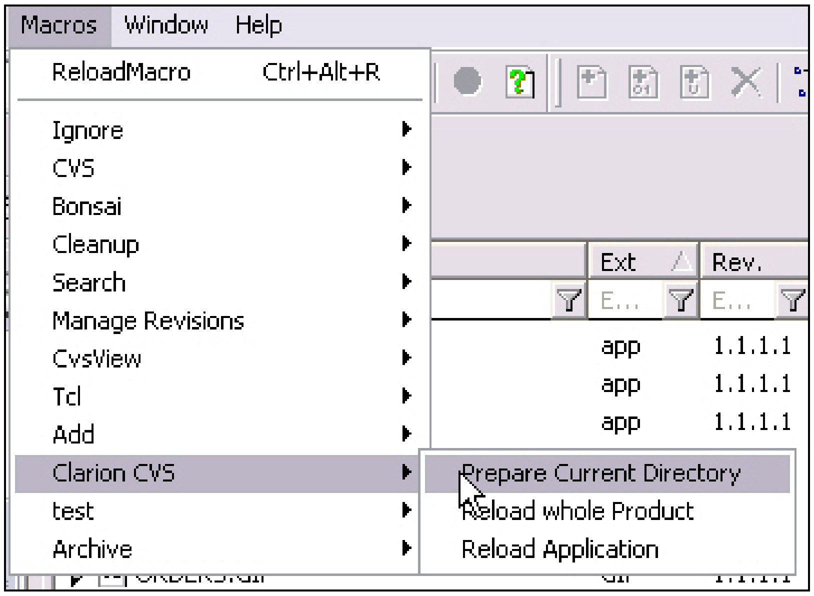
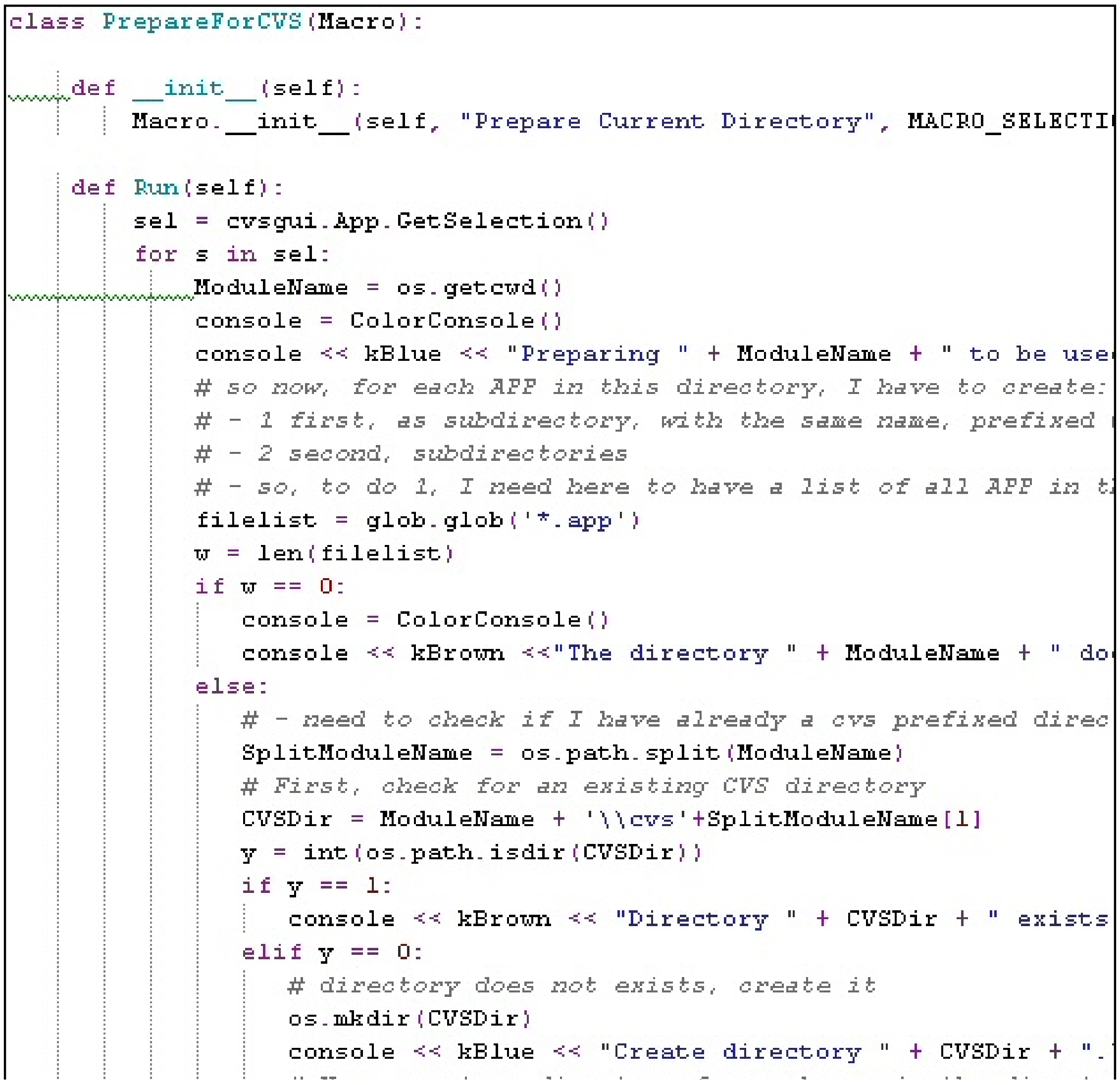


Figure 21. Macro to prepare directory selected from the Macros Menu.

You must have selected the directory you want to prepare, and this directory must contain at least one .APP file for this macro to do something.

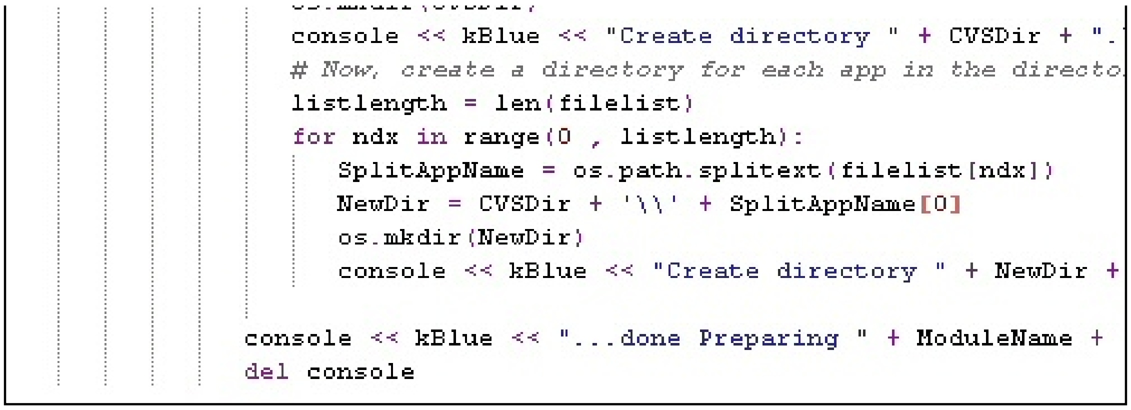
If you are curious about Python, Figure 22 shows the code for this macro:



```
class PrepareForCVS(Macro):

    def __init__(self):
        Macro.__init__(self, "Prepare Current Directory", MACRO_SELECTI

    def Run(self):
        sel = cvsgui.App.GetSelection()
        for s in sel:
            ModuleName = os.getcwd()
            console = ColorConsole()
            console << kBlue << "Preparing " + ModuleName + " to be use
            # so now, for each APP in this directory, I have to create:
            # - 1 first, as subdirectory, with the same name, prefixed
            # - 2 second, subdirectories
            # - so, to do 1, I need here to have a list of all APP in t
            filelist = glob.glob('*.app')
            w = len(filelist)
            if w == 0:
                console = ColorConsole()
                console << kBrown <<"The directory " + ModuleName + " do
            else:
                # - need to check if I have already a cvs prefixed direc
                SplitModuleName = os.path.split(ModuleName)
                # First, check for an existing CVS directory
                CVSDir = ModuleName + '\\cvs'+SplitModuleName[1]
                y = int(os.path.isdir(CVSDir))
                if y == 1:
                    console << kBrown << "Directory " + CVSDir + " exists
                elif y == 0:
                    # directory does not exists, create it
                    os.mkdir(CVSDir)
                    console << kBlue << "Create directory " + CVSDir + ".
```



```
    console << kBlue << "Create directory " + CVSDir + ".
    # Now, create a directory for each app in the directo
    listlength = len(filelist)
    for ndx in range(0 , listlength):
        SplitAppName = os.path.splitext(filelist[ndx])
        NewDir = CVSDir + '\\' + SplitAppName[0]
        os.mkdir(NewDir)
        console << kBlue << "Create directory " + NewDir +

console << kBlue << "...done Preparing " + ModuleName +
del console
```

Figure 22. Prepare for CVS python Macro source code

I am far from being an expert in Python, and I am sure this code could take less lines, but I always prefer my code to be readable. And Python code is very readable, due to the mandatory tabs. As an example, I will use the Clarion DLLTutor example, as this directory has four applications. Here is the console output in WinCVS once the macro has finished working:

```
Preparing D:\Clarion6\Examples\DLLTUTOR to be used with WinCVS and Clarion6.
Create directory D:\Clarion6\Examples\DLLTUTOR\cvsDLLTUTOR.
Create directory D:\Clarion6\Examples\DLLTUTOR\cvsDLLTUTOR\Allfiles.
Create directory D:\Clarion6\Examples\DLLTUTOR\cvsDLLTUTOR\Dlltutor.
Create directory D:\Clarion6\Examples\DLLTUTOR\cvsDLLTUTOR\Reports.
Create directory D:\Clarion6\Examples\DLLTUTOR\cvsDLLTUTOR\Updates.
...done Preparing D:\Clarion6\Examples\DLLTUTOR for CVS.
```

And here is what I get in WinCVS file manager once I refresh the display (F5)

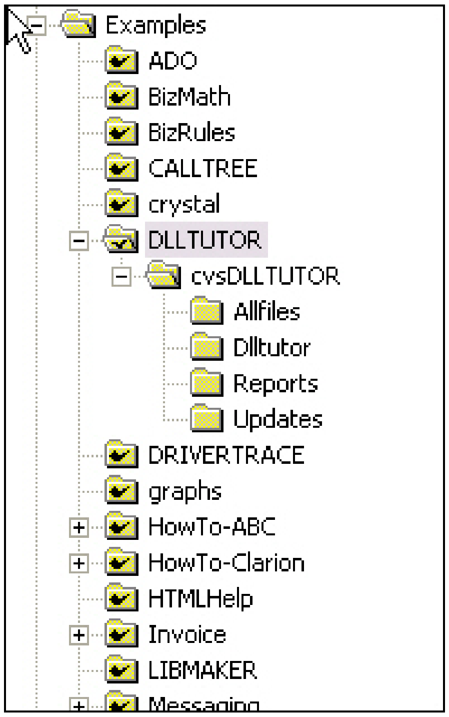


Figure 23. DLL tutor directory after the macro has been launched.

Reasons to use WinCVS rather than Clarion

The main reason to not interact with CVS entirely from Clarion is that Clarion works exclusively file by file. If you are a member of a development team, you need to periodically update your working directory, and it's a lot faster and easier to do so by updating the whole directory, rather than updating each module of an application. Imagine an application of about 25, 30 DLLs, one EXE, with an average of 25 procedures per application. WinCVS allow you to update all this source with *one click!* From inside Clarion, you will have wrist pain before being done with your 750 modules.

But, you are going to tell me, how can I use the APV's files that I just updated from my co-workers, without doing mouse gymnastics from inside Clarion?

As you have seen in the screen shot showing Clarion CVS macros, and with the example above, there are macros that I use to rebuild the TXA and DCT files from the APVs and DCVs files. These are in the downloadable zip at the end of this article.

Administration, automatisation

CVS works just fine without ever changing anything in your administrative files, but there are numerous cool features you can add to your CVS toolbox.

I talked a little about CVS administration earlier, when I suggested that you checkout your CVSROOT module, and update the modules file to be able to tag and branch your last release.

There is also a useful hook for a notification feature, by which CVS emails all the users when files have been committed to certain directories. CVS does not do this directly, it just provide hooks, in the form of files. These files live in your CVSROOT directory, and which you can use to run scripts. CVS reads those files, and executes anycommands they contain. For example, the loginfo script will execute *after* commit, once the log file is written, while the commitinfo script will execute *prior* to commit, to check that a commit matches certain criteria. (Note, all this works on Linux and *should* work on Windows, but it's possible some scripts may need an adaptation for Windows). You may also want to try Bo Berglund's CVSMailer script.

To get email notification to work, you need to use your loginfo file, found in CVSROOT, as well as a Perl script (part of the code to download with the paper). Here is a line of my loginfo file (line breaks added):

```
DEFAULT $CVSROOT/CVSROOT/mailer.pl -f
  $CVSROOT/CVSROOT/commitlog –m|
  bernard@bgsoftfactory.com %s
```

`DEFAULT` means "everything else", as in you can have other lines before, specifying other directories, and using other scripts. You can also have as many emails addresses as you want/need. `Commitlog` is the log written after a commit, so this line means that the Perl script receives that log as parameters after a commit, along with the specified email address(es) and the name of the files being committed. The script parses the log and sends an email to each address specified on the line,as this one I just received while making changes on Invoice.app:

```
Date:    Thursday February 10, 2005 @ 13:47
Author:    bernard

Update of /home/cvsroot/repository/Invoice
In directory linuxserver.bgsoftfactory.com:/tmp/cvs-serv30831

Modified Files:
      Tag: Branch_out_Invoice_for_Version_10-9
    INVOICE.app INVOICE.BPP
Log Message:
Change made on the branch.
```

This is very useful, mostly if you have more than one person working on a project. It helps to know who did what, and to remind you that you may have to update your code base before going to work on some of these files.

There are entire books explaining a lot more about CVS that I would ever be able to, and I suggest that you refer to them if you want to master CVS, along with the on-line documentation already suggested by Nardus. I have been using the following books to learn CVS and discover its possibilities:

- *Essential CVS, O'Reilly, ISBN 0-596-00459-1*
- *Open Source Development with CVS, Paraglyph press, ISBN 1-932111-81-6*

Tools, ideas

There is no real limit to what can be done to improve and extend CVS. For my work in Clarion with CVS, I have written a few macros in Python, as well as a couple of utilities in Clarion to be able to save time and make CVS a more pleasant and safe experience. You will find those in the downloadable zip at the end of this article.

One thing I have not done yet is link an update command on a directory with the macro reloading the apps in Clarion and a batch compile utility. That way I could update and recompile a whole product made of about 50 DLLs!

As of now I am working on a little utility which lives in the system tray, and grabs any email with "CVS update:" as subject. This way I avoid cluttering my mailbox, and I'm reminded of what I may need to update.

The administrative files in CVSROOT allow running scripts as "post-commit", and this could be used to clone your working directory with another directory to which each newly committed file would be copied. That would be an excellent way to have an always up-to-date copy of your work, on which you could periodically run a batch compile. As you can see, the possibilities are endless.

I hope this introduction to CVS has given you the desire to try and see for yourself what version control with CVS can do for you. If you have any questions, feel free to email me. I have also set up a page on my web site dedicated to CVS and Clarion. You can find further assistance there

Download the source

Bernard Grosperrin is a native of France, and has been a big fan of everything American since his teens. He began visiting the US in 1996, and moved to San Antonio in 1998, where he discovered his love of Mexican food. He and his lovely wife Gloria, who he met in Tulsa, now live in California. Bernard has been programming and designing software and databases for 14 years, primarily with Clarion. His hobbies include flying radio-controlled airplanes and riding his motorcycle. He loves aircraft of all kinds, and can't miss an opportunity to fly, whether it’s in a glider, a World War II trainer, or a general aviation Piper or Cessna.

Reader Comments

Add a comment

Clarion Magazine

Providing Good Customer Support

by Drew Bourrut

Published 2005-04-22

They call day and night, email you even on weekends, and actually expect you to help... customers! Arggh! But without them, you're out of business. So what is the best way to manage tech support? In this article I'll first define what I believe is good tech support, and then I'll suggest ways to assure that type of support.

Let's assume you have a product that requires you to answer questions about everything from perceived bugs in the product, to how to use some extremely simple feature. Let's further assume that you are interested in providing good support, both because you believe it's the right thing to do, and because it may help to increase business.

The customer's perspective

The first thing to do is to not look at support from your perspective but rather from your customer's perspective. What does the customer want? Above all, the customer would rather not have to contact you in the first place. So the easier your product is to use and the more bug-free and reliable the product the happier the customer (and having a too-buggy product could cause you to go out of business).

Second, try to anticipate your customer's preferred way of contacting you. Offer choices. Email and telephone are perhaps the most common. If the customer is going to use email, s/he probably doesn't want to fill out a complex web application form to ask a simple question. If by phone, the customer doesn't want to have to push a myriad of buttons before being given to the appropriate person. Further, customers don't want to be on hold for hours.

Third, if the problem cannot be resolved immediately and requires work on your part, the customer almost certainly wants to be kept informed of the status of the problem.

Finally, when the problem is resolved, the customer wants to know that the problem has been resolved.

Let me give a simple example. I have cable Internet access. Suppose one day I discover access to the Internet is down. I call my ISP and they say they will have to research the problem and will contact me. But I'm impatient, so after an hour I check... No service... Another hour goes by... no service... another hour... no service.

At some point I stop checking. Two days go by and I've heard nothing. Then I decide to check, and discover I have service. Should I be happy? I should be furious. When was it fixed? For how long have I had service and not known it?

How could this have been made better? My ISP could have said, "We will phone you when the problem is resolved." They then could have had some sort of automated system make the phone call and play a message stating that the problem had be taken care of. I would have been a happy camper!

So, when there is a problem that cannot be immediately resolved the customer probably wants feedback, and for such a customer probably no amount of feedback is too much.

Providing support

Okay, you're committed to good support, but it takes time to provide good support and you don't have the time. Simply put, this is nonsense. The key is in the methodology. You need to create a process that works for you, and then use that process each day every day, with every situation, big or small. For example, I use my own product (which I call PSI HD) to track problems and their resolutions.

My process

Every problem a customer reports to my company goes through the following process:

- Customer contacts me
- I enter the date and time of the contact, and a synopsis of the problem.
- I now either solve the problem or add it to the queue of problems to be solved.
- If I can immediately solve it I note what I did, complete the call or email, and then move the problem to the solved problem folder.
- If the problem must be added to the queue, I state how long it may take (is it hours, days, weeks?) and tell the customer.
- I keep the customer updated. For example, I have a standard form email letter I keep, and which I periodically send to the customer telling him that we are still working on the problem; in the email I also note any parts of the problem that have been resolved. It takes seconds to send out this email.
- When the problem is resolved, I call the customer if they've called me, or email them if I was originally contacted by email. I state the resolution of the issue.
- I note in the log what I've done, and then move the logged entry to the completed folder. By the way, this folder becomes a great resource for solving similar problems.
- Finally, regardless of the situation, after a period of time I double-check by either calling the customer or sending another email to ask if things are still okay.
- I periodically update the above process to make it better and more efficient.

The routine

The key to success is in the routine. Do it the same way each and every time, and always do what the customer requires.

Here's one more example. We were referred to a client (by one of our current customers) to solve a spyware problem. Simple, we went, we ran various pieces of software, and the spyware was reported as removed. We left a bill. The next day we called, and asked "Is everything okay?" We were surprised to discover that the same piece of spyware was back. We went back twice before it was fully removed. Time passed and that customer moved to a new office. We were asked to make sure all of the technology-related products were properly installed and set up at the new location. When I asked why we were being asked to provide this service, the customer said, "You're the first company that ever called us back after a service call to make sure things were working. And when they weren't you took care of it and checked back again. I know you'll do the right thing."

But let's take a deeper look at this idea of routine. Fast food chains know about routine.

They know that if you do it the same way each time you spend less time than if you re-event the wheel over and over again. You may see having a routine as something that diminishes creativity. I argue that's not so. Rather, having a good routine allows more time for the creative process. I have various routines and so do you. It may be getting up at the same time each day or eating dinner at a specified time. In general, routines are what make creative time possible.

Creating a routine

There are three keys to creating a routine that you *will* be able to follow.

- Put it in writing
- Automate as much a possible
- Re-work it when it fails to handle a situation.

Putting a routine in writing is key because a written procedure doesn't depend on your potentially faulty memory. Also, a written procedure can be revised. If you simply try to create a routine in your head you'll fail on two fronts. First, you won't be consistent, and without consistency you don't have a routine. Second, it's easier to more thoroughly re-work a written routine than to depend on your memory.

For example, suppose you create a routine for handling un-solicited calls. Occasionally you'll have a situation not covered by your routine. By writing down the routine you can easily modify it when it doesn't work.

Here's an actual example. When I receive an unsolicited call at home, I do the following:

- Ask the person's name and company, and write it down
- Note the date and time of the call
- Ask to be put on the "Do Not Call" list.

Recently a company has been calling my home with a completely automated message. I could not tell them to not call again since there was no person to talk to. But at the end of the message there were instructions to press the numeral two to be put on their Do Not Call list. I pressed it. A week later I got another call from them and I had to listen to the whole message again to learn what number to press to remove myself from their call list. But this time when I hung up I changed my procedure and noted it on my form. I then updated the written procedure to remind me and my wife of this change. I haven't stopped them from calling but now I immediately hit 2 and hang up.

Once you have a routine, automate it as much as possible. Our company has a holiday card list, which I maintain using a piece of software that I use for no other purpose. For quite some time I had a problem remembering what software to use and how to use it. I created an automated procedure that, unasked, prints a document December 1st. This document reminds me of what I need to do and how to do it. Shazam!

Finally, rework your routines. As you saw in the phone example I changed the routine when it failed to work. And make a routine of revisiting your routines. You'll only make the process better.

More on Automation

Here are three possible scenarios where automation can really help: a hardware manufacturer, a software manufacturer, and a tree spraying company.

The hardware manufacturer

Let's say you build computers that are sold over the internet. You probably get two types of support calls: "How do I?" and "It doesn't work!" calls. The "How do I?" calls are the easiest to automate. Every time you get a question, you enter that question into your support log. Then you log the solution into the same support log, and finally add the question and solution to an online knowledgebase searchable by your staff as well as by the customer. Don't just rely on the Internet version - by having an offline log you have access even when your Internet access is down. The next time similar a question is asked, the staff already have the answer. Further since the customer can find the answer online, they may not call.

Hardware problems can also use automation up to and including issuing a Return Material Authorization for computer return. But there is more you can do. When the computer is received your system can automatically send an email to the customer stating that you are in possession of their machine and telling the customer when they can next expect to hear from you. When a tech puts the hardware on the bench and starts to look at the machine this can trigger another automated email, and every time the tech notes what she is doing to solve the problem that can also be emailed to the customer.

The customer who knows where things stand is less likely to be calling you and complaining.

The Software Manufacturer

Suppose you make a software product that you sell on the Internet. You know that when you release version 1.0 that there are probably bugs in the software, and certainly there will be updates and upgrades. So you're not going to be surprised when a customer emails you saying they've found a bug. You can automate acknowledging the receipt of the message. You can automate making sure the email is passed on to the correct person in your organization. And you can automate keeping the customer updated on solving the problem. Sometimes the customer may find a way to work around a bug. You can encourage the customer to report these workarounds. Then finally all of this information can be automatically added to an online searchable knowledgebase.

The Tree Spraying Company

I have a customer who has a tree spraying company. We've created custom software for them that not only handles their sales and accounting functions but also helps them automate handling complaint calls from their customers.. Typically, they get complaints that a spraying was not done when it should have been, and complaints about the quality of work. Perhaps they sprayed right after a rain storm and the customer wants a re-spraying. They have a routine for handling both phoned-in complaints as well as emailed complaints. They also could have an automated, online system that would allow their customers to reschedule a missing or poorly done spray, and also find out more about tree spraying in general. There they might learn that an hour after a rain storm is a great time to spray against insects.

Summary

I've discussed those issues directly related to support from the customer's perspective. Other things to consider include tracking your time, tying support and billing together, and even handling issues of replace vs. repair.

What's important is that we can all do better at supporting our customers. Routines are essential, and finding ways to automate those routines is just as important! You already know that keeping the customer happy is good business and besides, that customer may be me!

Reader Comments

Add a comment

- » This is a very helpful article for me. I write software...

Clarion Magazine

Planet Clarion Transcript: Clarion.NET, and Trial Versions

Published 2005-04-25

Dave Harms: This is Planet Clarion for March 30th, 2005. I'm Dave Harms.

Andrew Guidroz: I'm Andrew Guidroz.

Dave Harms: Good day.

Andrew Guidroz: Good day.

Dave Harms: On today's show, we're playing Part Two, we're playing the secret tapes of the conversation that I had with Bob Zaunere. Prior to the conversation that we played last time - - is that too confusing?

Andrew Guidroz: No, I don't think so at all. It's sort of the prequel.

Dave Harms: It's the prequel, exactly. That's just the word I was looking for. Today we play the prequel. This is Star Wars One.

Andrew Guidroz: That's right. This is when you get to find out exactly who Bob Z. was before he put on the black mask and the helmet. Bob's going to cover some of the things that happened before I got in the middle of the conversation. So it's going to be primarily Dave and Bob. We'll do the take.

Dave Harms: Okay. In fact, it'll be exclusively Dave and Bob.

Andrew Guidroz: Exclusively.

Dave Harms: Yes.

Andrew Guidroz: Of course.

Dave Harms: All right, roll the tape. Let's roll it.

<start of interview with Bob Z>

Dave Harms: How about if somebody wants to - can't really wait for the subscription program and they want to cut their teeth on .NET? What's the best way to do that? Should they be looking at getting C# or something like that?

Bob Z: Well, I guess I have two comments. I don't think that C# is really the answer and - although I think it is a great language - really, it's pretty nice. But, at the end of the day, it's a language at the level of C code. So it's entirely different from what we write in Clarion. I think I mentioned, and it was often misunderstood, but part of the approach that we took to doing the .NET compiler was to ensure that we could compile the same code in C#, and compare the output and be able to debug either assembly, which is what they call DLLs and EXEs in the .NET world, and to ensure that we could get equal performance to anything done in C#, and equally to ensure that we didn't - that we followed the right paths and we're following what Microsoft intended the language - intended the platform to do, in the same manner. But what we've come to find out, not - no surprise there, but you might write five lines of Clarion code and then when you get the equivalent C# code, you might be looking at 150 to 200 lines, to do the same thing, particularly when dealing with file structures.

Dave Harms: Yes, I think we Clarion developers are a little bit spoiled when it comes to file handling, that's for sure. So when are we going to get to see Clarion.NET?

Bob Z: The people who really are anxious to get into .NET and Clarion.NET probably won't have more than a few weeks to wait before they will be able to do so. And that would be via the subscription program. The other thing is that Bob Foreman, and a couple of others, have been hard at work on the beginnings of a whole .NET training course, which will both introduce .NET as a platform, and what you have to know about it, and, of course, from that platform lead you right through how to work with Clairon.NET. And that'll also be available, I don't think more than 3 to 4 weeks out - the first of it. I mean, this won't be - it won't be complete because to cover everything in .NET, you could easily spend, I don't know, a few months, and you'd be working pretty hard at it. Obviously it's a big - there's a huge class library behind it.

Dave Harms: I know, and obviously Microsoft thinks that .NET is the future. But what's in it for Clarion developers? What's going to motivate people to start getting involved with Clarion.NET now, as opposed to waiting a couple of years down the road?

Bob Z: Part of the problem that we wanted to address, really, is that it's not for everyone, right now, to say, "You know what? I'm going to call this API call." Or, "I want to use this C++ library." And there's some intricacies, depending on the API, or the library that you're working with, that go beyond what many people want to work with, or that is easy to work with in the sense that it requires a lot of knowledge that you don't get if you haven't worked in that world. If you haven't worked in C++, then you're probably not going to feel comfortable working with a lot of the API calls, for example. We wanted to try to make that not be the case with the .NET class library. We wanted to make it very, very simple - so much so that you should be able to pick up any example code and go, "Oh, okay. I'm just going to write that using Clarion syntax, to do the exact same thing." And right now it's kind of hard, for example, to go - a lot of guys go, "How do I make this VB.NET code do the same thing in Clarion?" And it's not easy. I read an article, not too long ago, maybe a few weeks ago, that said, well, to set up a few classes to work with ADO - not ADO.NET - and to take the NorthWind example, it would take 60 pages of code, and with .NET, it's going to take 140 to 150. But, if you think about it, for Clarion, it's going to take, I don't know, two pages of code. When you think about it, it's true - three pages. And that is going to be true in Clarion.NET as well. So the point being is that it would not be - it's not a big leap to think that someone may look at Clairon.NET and go, "You know, this really is going to make my database work much easier. But I have all this other code and it's in VB or it's in C#." And you'll be able to easily export the assemblies and work with them from other languages, unlike - it's a little difficult right now.

Dave Harms: Yes. It just opens up so many avenues, doesn't it?

Bob Z: Yes, it really does.

Dave Harms: It's quite something. So what is the status of Clarion.NET right now?

Bob Z: The Compiler project has just really made excellent progress. There's always, simultaneously, there's the RTL, the critical parts of it are being "ported." There's some decisions left to be made as far as file drivers and file systems, particularly for ISAM files, which aren't well supported in .NET - obviously none of the proprietary ones like TopSpeed and Clarion files are. There's a lot to be said about the ADO.NET provider for Btrieve, and whether it's any good or not so good, or if to make it Access is better. And so there's some work to be done and some decisions left to be made in that area. And the window handling is a big, big thing. We tackled WinForms first, because WinForms is the .NET way of doing Windows - period, across all languages. And we intend to support it completely. And the implementation of the Clarion window handling will, of course, be built on top of WinForms.

Dave Harms: So when people get this first look, they'll basically be able to - it'll be kind of like, I guess, maybe the first alphas of Clarion for Windows, where you could do some fairly basic stuff, but there will still be - or are you trying to get a lot of the RTL ready for that?

Bob Z: Well, a lot of the - the file handling will certainly be ready. The windows will probably have to be WinForm windows initially, for the very first release of it. So I wouldn't really characterize it as you'd only be able to do basic stuff. You'll have less versatility maybe, or more code to write is really what it boils down to, for window handling.

Dave Harms: Yes, but you'll be able to start playing with a new IDE and you'll be able to…

Bob Z: You'll be able to write .NET code.

Dave Harms: How much of the IDE will be in that initial release - the new IDE?

Bob Z: Good question. Most likely AppGen will not be in a state that I'll release it during that time. It's possible but I don't think it will be. The new project system will be - which I don't know if we ever talked about that, but I think I mentioned, I did mention it at DevCon and talked about it a bit - the solution based system. The fact of the matter is that that will be kind of an interesting thing because in Clarion 6, at least, it's very simple to go and export your projects, from your app files, for example. And you're going to be able to build these solutions around them. And the project system has really been - well, it's really being brought up to date in the sense that the Topspeed project system, has always had incredible capabilities - much more programmable than, say, Visual Studio .NET even. You can just do some really neat things with it. Those features or that power was never brought into [the] Clarion IDE, period. It will be part of what will be supported with the new IDE. So that will be there. The new editor will be there, with all of its features. I'm not sure the code completion will be there yet, because we're not doing simple code completion. Code completion, really there's two ways to do it. There's the alphabetic look-up, which is I guess base-level code completion, where you just complete a - you just have a database, and if someone goes, A,B,C-dot, and you go, "Okay, let me go find A, B, C, or A-B type, and you're just doing a look-up. There's no syntax analyzer. So there's nothing that really understands the code. A true code completion system has a syntax analyzer in it. So it understands what’s a class, what’s a queue, what’s a file. It knows about driver classes. It goes much further. So I don’t think - the end-goal for that, won’t be in the first release but it will most likely - some of the code completion has already been plugged in, some of it’s being completed. So some code completion, the code folding will be in there. And there’s some really cool stuff - some new to-do lists. There’s a lot of stuff that’s going to surprise people that they’re going to really like in the new editor. Yeah, it’s going to be good. It’s a nice implementation. And then the dictionary, I’m not sure. The dictionary has evolved from what I showed at DevCon - quite a bit, by the way. And that could be a topic we could talk about in one of the interviews because there’s been some additional things added in. There was some things I brought up at DevCon that people expressed real interest in. And so we’ve re-factored some of the code and we’re making allowances so we can support those things - example being multi-user dictionaries. So that’s going to be some neat things too.

Dave Harms: But if I understand you right, it sounds like with the new Clarion - Clarion 2005, whatever it’s going to be called - that the first release of that - because if the AppGen stuff is not really quite ready, the first release of that probably will be more of a .NET supporting release than a Clarion supporting release. Is that accurate?

Bob Z: Well, I think I would characterize it a little differently. I’d say that if AppGen doesn’t make the cut, then it’s obviously more of a hand coder’s or - release and/or, if you’re willing to take the time to just export your projects, assuming they’re app-based, then it will be able to be used immediately to build solution files through applications. So you can go off and go, "Hey, build me this solution. Build me Debug. Build me this in release and this in Debug. Build me both." And so you’ll be able to go off and you’ll be able to do the whole thing without - I mean, there’s a lot of good compile managers out there, but they rely upon DDE. They have to. And, as we know, XP has pretty much done away with DDE support, and just about made it useless, in many cases. So, it’ll be a lot better. It’ll be markedly faster for building large systems. There’s no question. And dictionary [editor], like I said, it may be ready by then, at least an early cut. It’ll probably be ready to look at. I’m not sure how it would--. I will be suggesting people to migrate the dictionary work over. But that’ll be an interesting thing. There’s some interesting things, with the move to .NET. I don’t foresee people just going, "Okay, all my development is in .NET." Now some will, obviously. They have external pressures and it’s, "This is what corporate said or this is what my customers want, and that’s all I’m doing now." But, for most people, they’re going to need to live in both worlds, for quite a number of years to come.

Dave Harms: Yes, I think that's a reasonable expectation. So what comes first? Clarion 7 or Clarion.NET?

Bob Z: Well, there's two different teams working on both. So it's not going to be an either/or. The work on the IDE is central to us going forward. So that is really the prime focus, as an end-goal to get something into people's hands. We want people to get into the .NET compiler, early on, so we get feedback, so we can refine the manuals and the training materials, and to deliver what people - those people who say, "I have to have .NET, a month ago - I needed it" - to be able to start to work with it right away. But we don't - it's not a case where we have to choose, well, should we move people off of that, onto this, because there's been a clear division, right from the beginning, on who will work on what. As we get closer to both projects, getting to their goals, then there's going to be - then there'll be some shared work, because the - some of the integration for .NET support into the IDE will require people from both teams to work on it. And the same for some of the run-type library support and the driver support.

Dave Harms: Let's, for the sake of argument, call the next version of Clarion, Clarion 7. So if I've got a C7 app and a C6 app open in the new IDE, when I switch to the C6 app, does it look like - am I looking at the same kind of prompts and everything, as I see now, because obviously the templates are going to be the C6 templates?

Bob Z: Yes, it's going to be interesting. You will see everything wrapped in the new look and feel of the IDE, which has changed, but your prompts will be identical, to what you see in Clarion 6.

Dave Harms: What about the sort of jumping around business where you open this window, or will it flatten out at all, or how does that hardware--?

Bob Z: Yes, the IDE won't change in its behavior. Aside from the prompts you'll see - based on templates, because those prompts come from the templates. But other than that, the IDE will be the same, whether you're in Clarion 7 or Clarion 5.

Dave Harms: Wow. Yes, as long as we're on the subject of templates, I wanted to ask you something about DevCon. Remember you were asking for a show of hands on some new features, and you asked whether anybody would like compiled templates, and a voice from the back said, "As long as we can decompile them."

Bob Z: Yes, it's funny to hear them say that, because you think about it. Right now, there's 3rd party utilities that have to ship as a DLL, to protect their intellectual property.

But people should be more concerned about that than they are about can I decompile the template - because if you can't write a template, you're not going to need to decompile it and work on it and recompile it. The chances are, you report a bug to the 3rd party guy and he's going to fix the bug - not to be very - so I was kind of surprised at that. I thought 3rd party people would go, "Jeeze, that makes my life really kind of easier, because I--."

Dave Harms: Yes, I have to say that I still sort of wonder about that one because--.

Bob Z: Yes?

Dave Harms: Yes. When I heard the comment, I thought - I expected the comment. I mean, I wasn't expecting it, I wasn't surprised by the comment. And I think that maybe is a different perspective. It's something that maybe you don't see as much or get a sense of as much, as the vendor of the language. But out there, it does seem to be that there is - if anything, there's more of a trend towards openness, where you get the template source with it, or you get the source code with it. And that seems to be more of a driving force than the ability to protect intellectual property, in a way. I see the value to the 3rd party vendor. The question is - it's not a question of value to the 3rd party vendor. It's a question of the perception on the part of the user of that product, whether they're--. People, I think, are always a little bit leery - have become more leery, over the years, about black boxes. So when people have DLLs and source coded stuff, then they create a version that [has] no more black box. You know?

Bob Z: Yes.

Dave Harms: And they're doing that because there's a commercial pressure to do that, I think.

Bob Z: No, I would agree with you on that. I tend to agree on that. I don't see our template chains, ABC and Clarion chain, shipping in compiled form. However, that doesn't preclude us from producing a student version that would ship, with a given compiled template chain.

Dave Harms: And that I can see as a really good reason for having that ability.

Bob Z: I would think 3rd party people would go, "Jeeze, I could do the same now. I can have a demo version - It's compiled.

Dave Harms: Yes, I could see that.

Bob Z: And if I - you get the source code of the template.

Dave Harms: I think that would be a wonderful use of that.

<end of interview with Bob Z>

Dave Harms: And that's my conversation with Bob Z.

Andrew Guidroz: It's sort of like a conversation with me in the room with a gag on.

Dave Harms: Yeah, yeah, we ought to try that sometime.

Andrew Guidroz: Yeah, I appreciate that. About a trial version of Clarion…

Dave Harms: Yes.

Andrew Guidroz: Why are you so gung-ho about that?

Dave Harms: Well, I guess that probably just for the sort of nonsensical reason, that I've used trial versions of many products and it's often helped me to make a decision about whether or not to buy it.

Andrew Guidroz: Did you use that to get into Clarion to start with?

Dave Harms: At the time, I don't think it was very common to have trial versions - well, you didn't have downloadable trial versions of things - but I certainly would obtain trial versions of a few other products at the time I was looking at Clarion, yes.

Andrew Guidroz: Can you think off the top of your head the last software product that you bought? Something relatively recent that you purchased - did you try it in trial first?

Dave Harms: Yeah…

Andrew Guidroz: That's the question - right?

Dave Harms: FrameMaker. Sure, definitely trial version first. I think basically any - it's really - it's a lot harder for me to make a decision on some software if I can't actually try it out.

Andrew Guidroz: Is FrameMaker expensive?

Dave Harms: Yes, FrameMaker's around a thousand bucks or something like that, I think?

Andrew Guidroz: So maybe the key to the trial version is the fact that the cost of entry is high…

Dave Harms: Yeah.

Andrew Guidroz: And that's what brings you in. Because what I was thinking of is the fact that it's been awhile since I got a trial version of anything. But when I look back at what I've done - lately I've gotten into digital photography. So I picked up a copy of, what is it? Photoshop, what's the geared down version? - Photoshop Elements. Okay?

Dave Harms: Mmm hmm.

Andrew Guidroz: And I bought version 2.0, which was the current version, at the time, that I was doing my thing. And it only cost like about 69 bucks, with the rebates.

Dave Harms: Yes, and if it's 69 bucks and you have a fairly good--.

Andrew Guidroz: The cost of the entry was low.

Dave Harms: You have a fairly good idea that it's going to do the job, you don't have a problem, because it's an impulse buy.

Andrew Guidroz: And it was okay.

Dave Harms: Kind of like Clarion Magazine. You see with everything - "Yes, that'll do the job." Impulse buy.

Andrew Guidroz: There you go. But this thing here, it was cheap so I bought it. It didn't do exactly what I wanted it to do but it was - I didn't feel ripped off.

Dave Harms: Yes, now if that had been a 600 dollar product or a 2000 dollar product.

Andrew Guidroz: I'd have felt ripped off. Yeah, I'd have wanted to test drive first. For sure, I'd have been very aggravated afterwards, because it didn't do exactly what I thought it was going to do. But, what ended up happening was 3.0 came out, like right about the same time. And I'm wondering if the discounting of 2.0 was done on purpose, because they knew that 3 was coming and it would try to drag you in. And 3.0 was awesome, in Photoshop Essentials.

Dave Harms: And this is relevant to trial editions?

Andrew Guidroz: Yes, I think it is. I think that--. Do you like the product you were just taking about? FrameWork?

Dave Harms: FrameMaker.

Andrew Guidroz: Yes, FrameMaker.

Dave Harms: Well, FrameMaker is well sort of the lesser of all evils, when it comes to publishing books. But it's very old. It does the job. It actually does the job much better, for the kind of stuff I do, better than many newer tools.

Andrew Guidroz: And you like the product.

Dave Harms: Well, it's certainly useable and it does the things that I need. It's really showing its age.

Andrew Guidroz: But it's value for the money?

Dave Harms: It's value for the money - for sure.

Andrew Guidroz: Yes. And I think that's what people are looking for. And I think that's the thing with the trial. So maybe what we're talking about right here isn't relevant to the normal listeners. Maybe it's more relevant to Bob Z. Maybe it's like, hey Bob, this is why we think - your products that are expensive, get trial versions out there because we kind of like to poke it before we - poke, kick the tires and take a look around at this thing.

Dave Harms: Yes, I think there are trial versions of - I've looked at all kinds of development tools - something like Rational Rose, a modeling tool. You can get a trial version of that, or at least you used to be able to. You can get trial versions of all kinds of stuff. It's pretty common. And I think that that would really be a big help. And the whole business of compiled templates certainly makes that a lot more practical because you don't have to worry so much about giving away the--.

Andrew Guidroz: Right. Oh, I see what you're saying. With the compiled template set, a compiled template set would be useable for people that want to sell test drives.

Dave Harms: Yeah.

Andrew Guidroz: For, what's the word, trial versions.

Dave Harms: Well, that's what Bob was saying was that it could be useful for 3rd party products. I mean I still maintain that--.

Andrew Guidroz: Yes, well when he said 3rd party products it - that didn't click. I understand the concept that they want to--.

Dave Harms: No way, and it doesn't--.

Andrew Guidroz: Well, they want to protect their investment in writing templates. I understand that. I understand that some people want more closed deals. Some software people - there was talk about, what was it? Decompiling templates.

Dave Harms: Yes.

Andrew Guidroz: Look. I don't think that should be a consideration at all, by Soft Velocity. The fact that some users are nervous that some compiled templates will certainly start happening out there. I think compiled templates are fine. I don't see a problem with it. I think that software developers are mature enough to know when they should expose things. Heh heh. Of course, I'm not mature enough to say it without giggling. I think they're mature enough to know when to expose things and when not to expose things. Right? If the market's screaming for source code, and the market is willing to pay for source code, then these guys, they're going to sell source code.

Dave Harms: Yeah.

Andrew Guidroz: The money's there. And the same thing with the compiled templates is that if the market's not there, it's just - it's going to disappear. The market will dictate it.

Dave Harms: And I think Bob's point was that even if the 3rd party developers don't want to - even if they want to sell product, they want to sell the source code, the template - if they want to release template source code, they certainly can - or at least if they've got the compile option, they can release demos, of what the templates do.

Andrew Guidroz: Sure.

Dave Harms: And they can put - because they're compiled, you can also put in limitations in those templates and then somebody can't go and remove the limitations, just by editing the source code. So it'll actually raise a whole 'nother issue of how secure the compiling is, and whether it's - people bother trying to find a way to decompile it. But obviously you can build some sort of limits in or you can have your templates always generate some, "Trial Version", "Demo Version", into any reports or browsers or anything like that.

Andrew Guidroz: Exactly. And is this confession time? Should I come forward?

Dave Harms: Oh, well if you think that this is--.

Andrew Guidroz: I have taken 3rd party templates and read them.

Dave Harms: You rascal.

Andrew Guidroz: And said, "Man, that's a clever idea, the way he's doing that to do field assignments, in this little section. I think I'll use that in some of my own templates."

Dave Harms: Yes. Of course, you're not selling those templates, you're just--.

Andrew Guidroz: No, I'm not selling them. And I look at those guy's code, and I learn from them. The way I learned how to write templates was reading the standard templates - just reading through them and seeing what Soft Velocity did or I guess he who shall be nameless, the original owner, the original company that had it. That's how we learn. And I understand compiled templates may take some of that away. But I think the market will figure it out. I have the utmost confidence that when some vendor has done something no one likes, we drive them out of the town.

Dave Harms: Right. Run them out on a rail.

Andrew Guidroz: There you go.

Dave Harms: Covered with tar and feathers.

Andrew Guidroz: Tarred and feathered.

Dave Harms: Is that how it’s done? Are those separate things? Riding someone out of town on a rail? Would they be tarred and feathered at the same time?

Andrew Guidroz: No, by the time you tar and feather them, there isn’t a lot left. If you’ve seen - what’s the HBO series, the brand new one? Carnivàle, a few weeks ago.

Dave Harms: No, I can’t say I’ve seen it.

Andrew Guidroz: A gentleman was tarred and feathered and, believe me, boiling tar, does not feel good.

Dave Harms: Oh yes. Okay. All right. Well, this is a family show.

Andrew Guidroz: Oh, I’m sorry.

Dave Harms: So, we won’t maybe go there.

Andrew Guidroz: Oh Brother Aren’t Thou, there’s a scene where they run the opposing political candidate out on a rail. They literally throw the doors open and come in with a large wooden rail, pick him up, set him on top of the rail, and run him out. It was pretty good.

Dave Harms: That’s always the mental picture I’ve had.

Andrew Guidroz: Yes. There it is.

Dave Harms: Yes.

Andrew Guidroz: Okay. Now, we got one other thing. I have one other topic I want to squeeze into the broadcast this week.

Dave Harms: Fire away. Another confession?

Andrew Guidroz: Well, I have a coffee mug, that I use a lot, and it’s got a company name on it - well, it’s he who shall remain nameless, on my coffee mug. Bob Z., by now it’s time, now that everyone’s screaming for this and screaming for new features...

Dave Harms: It’s time for new coffee mugs?

Andrew Guidroz: I need a Soft Velocity coffee mug. And I've asked about it. In the past there weren't any yet, the last couple of times I've asked. And I think it's a cop out. I think that there really are coffee mugs there, with Soft Velocity written on it. Just set a price. You don't have to market it. I'll call. Just drop me an email and say, the Soft Velocity coffee mugs are available. I think that there's too many coffee drinkers in the Clarion community.

Dave Harms: Yeah.

Andrew Guidroz: Sell them 10 bucks a pop, if you got to. Just go for it. Go for it. We need a Soft Velocity coffee mug to go with our new .NET. That's what I have - that was very important to me today, to get that out.

Dave Harms: Do you feel better now?

Andrew Guidroz: I do, yes.

Dave Harms: Yeah, I can tell. Would you settle for a Clarion magazine coffee mug?

Andrew Guidroz: Clarion magazine has coffee mugs?

Dave Harms: Well, this is possible.

Andrew Guidroz: If you could get one, now that I'm crying for one. Why not?

Dave Harms: For the right price. Yes. I think they're probably available, for the right price.

Andrew Guidroz: Clarion magazine on one side and what? - the podcast logo on the other?

Dave Harms: Hey, hey, I think maybe we ought to do that. Yeah, yeah. I like that. Okay. All right. Well, send your orders, or send your interest. If you're interested in any Planet Clarion coffee mugs.

Andrew Guidroz: With the podcast, on the back. Oh, and we can--.

Dave Harms: Yes. A complete podcast, with every coffee mug.

Andrew Guidroz: Yes.

Dave Harms: What you do is, you hold the mug up to your ear and you just - you listen, very carefully. Okay. Well, I think we're done.

Andrew Guidroz: We'll have to call it a day.

Dave Harms: We'll have to call it a day. We got to get to get this one out.

Andrew Guidroz: Because we can't call it a night. That always throws you, doesn't it?

Dave Harms: Yes, because if we do it late enough, we can call it a night.

Andrew Guidroz: Can we go on at night? Let's call it a night.

Dave Harms: Let's call it a night.

Andrew Guidroz: Goodbye Dave.

Dave Harms: See you later Andrew. Okay. Bye.

Reader Comments

Add a comment

Clarion Magazine

DLLs and Reusable Code: Divide and Simplify

by Jeffrey Slarve

Published 2005-04-29

Many years ago, when I was relatively new to Clarion for Windows (I think it was version 1.0), I remember reading an essay by Dave Harms about splitting an application into DLLs. It was a big jump for me to grasp why splitting an app up was a good thing, how dingdang easy it was to do, and how much faster my development progress would become. But get it I did. DLLs allowed me to work on small chunks of a project without having to re-compile the entire thing with every little change/test that I needed to make. All was blissful in the world, and I lived happily ever after. The end.

Well, not quite the end. As time went on, and I had project after project (project, in this case, meaning all of the components of a software application) to write, I found myself duplicating code between projects. Many of these functions were very generic and didn't rely on a dictionary, so I tried making a generic DLL that did not require a dictionary or other global data, but the Application Generator thought it was way smarter than I was (which, in fact, it might very well have been). The AppGen wanted to either generate a bunch of global variables (`GlobalRequest`, `GlobalResponse`, etc.) as External, or Local. If global data was Local, meaning memory would be allocated for these variables in the DLL, then the templates would generate the .EXP file and include those global variables, causing duplicate symbol linking errors. If I told the AppGen to make global data external, then I had to make sure that any global data used by my DLL was also declared in any project where I wanted to use my generic DLL.

At first, I tried tricking the Appgen into doing my bidding, using the Local setting and modifying the .EXP file after the code was generated, but eventually I gave up with using Appgen to create a generic DLL. It was way too complicated. Especially when ABC came to be, and there were all those global class declarations.

But yikes! Now what was I going to do? Creating a DLL by hand was an option, but that seemed complicated, and way too much trouble. But then I remembered that the code would rarely change, so this effort would basically only have to be done one time and it wouldn't hurt to learn how to do this. Besides, I had much to gain by putting my frequently used generic functions into their own DLL:

1. **Reusable code**. No more rewriting/importing a `DayOfWeek()` function, or a function to get a widget from a dofladgey. All that has to be done is attach that special DLL to an app and it's ready to go.
2. **Write once**, modify in one place. If any changes need to be made to any of the generic functions, then they only need to be fixed in one place. Often, the applications that make use of this generic DLL will not need any recompiling. They only need recompiling if one of the following situations occurs:
    a. Any of the prototypes change. If any of the prototypes for the functions/procedures in the DLL change, then the applications making use of this DLL won't be able to find the procedure unless they are recompiled.
    b. New procedures are added. It's usually safe to not re-compile an existing app if all you did was add some new prototypes to the existing generic DLL. If, of course, you need to make use of the new functions, then a recompile will be needed regardless.
    c. New Clarion version. You can't normally share a DLL that was compiled with one version of Clarion with another. Well, you can, but not without a lot of knowledge, or special tools. It's better to recompile.
3. **Easy maintenance**. Once you set a DLL up, it's very easy to make changes to it. It's way easier than trying to make these changes to the same DayOfWeek() function in 20 apps.
4. **Coolness**. DLLs are very cool. Very very cool, indeed.

Handcoding the DLL

There are a few things you need to create when hand coding a DLL. Some of them might seem a little bit foreign to someone that hasn't dealt with this aspect of programming before, but they really aren't difficult to do once you get the hang of it. These steps are as follows:

1. Create a folder
2. Create a PRJ file
3. Create the source file(s)
4. Create an EXP file

5. Compile the DLL
6. Share the DLL prototypes
7. Make the DLL available to other applications
8. Use the DLL

Create a folder

The first step is to create a new folder somewhere in your Clarion development directory tree. It's a good idea to give your DLL a special place for all of it's components. It's the least you could do for it, seeing how well it will treat you in the future.

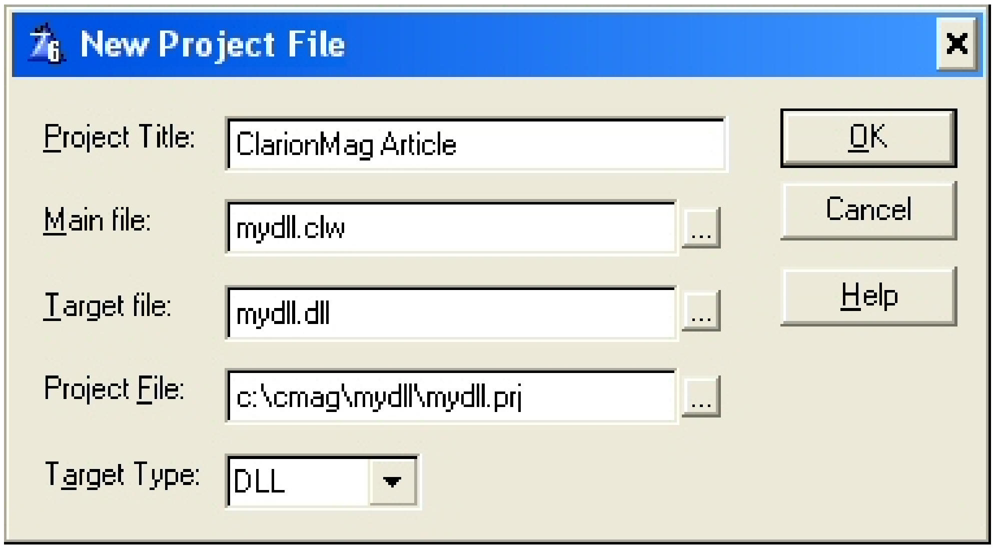
Create a .PRJ File

The .PRJ or Project is the daddy (or mommy, if you prefer) of your DLL. It needs to know everything that's included in your DLL, including file drivers, external LIB files, icon resources, and source code. Without the project, I don't know what I would do. Probably go fishing or play guitar.

Creating a PRJ is very easy in the Clarion IDE:

a. Click File|New then select Project.
b. You'll be presented with a file dialog where you can type the name of your project. This may or may not be what you choose to name your DLL. Click Save.
c. Now you should see a window that says New Project File. In this window, you can:
    i. Enter a Project Title. This field is optional.
    ii. The Main File is the name of the main .CLW file for your project. Just type the name here, then press TAB. It needn't be a file that exists yet.
    iii.

The Target File will probably be the same as the name of the Main File, except it will have a .EXE extension. This will change to DLL, after you change the Target Type to DLL. Click OK.



Project Title: ClarionMag Article
Main file: mydll.clw
Target file: mydll.dll
Project File: c:\cmag\mydll\mydll.prj
Target Type: DLL

d. You have just created a project. If this is your first project, then congratulations are in order.

You might have noticed that there is another Target Type called Library. Libraries are very similar to DLLs, but they have advantages/disadvantages when compared to using a DLL. I prefer to use DLLs because they are easier to manage, and require far fewer full-rebuilds of all of your .APP/.PRJ files than do .LIB files. If you make any change at all to a LIB, you will need to recompile (well, at least relink) all applications that use the LIB. The only downside to a DLL that you have one extra file to distribute with your application. Usually, having that extra file is an advantage because if you make a change to the DLL, all you have to do is ship that one file.

Create the .CLW file(s)

None of this would go anywhere without your .CLW or other source code files. These are where you write your functions/procedures. Just for the purpose of this article, I'm going to use one of the functions on my ClarionFAQ.com website. This function is used to move a Parent control (`GROUP`, `OPTION`, etc.) on a window, and subsequently move all of its child controls in pseudo unison. If you'd ever tried to move one of these parent controls at run-time, you'd know that the child controls have no idea that they're supposed to move along with their daddy too.

There are a few minimal things that need to be placed in the main module of your DLL's source code in order for it to compile.

a. `PROGRAM` or `MEMBER` statement. Even though this DLL isn't a program or a member of anything yet per se, the compiler needs this. If you use `MEMBER`, be sure to use it without any parameters. This can go on the first line, anywhere after the first column.

b. `MAP` statement. This is where you put the prototypes for the various functions that you'll be using in and exporting from your DLL.

```
map
   JSMoveParentControl(Long pParentFEQ,Long pXShift,|
     Long pYShift,Byte pMoveType=0)
end
```

c. `CODE` section. This is where you, uh, write your code. Please see the accompanying mydll.clw to see how this all goes together.

Create an .EXP file

The .EXP, or Export or Module Definition file is what you use to tell the project what variables and/or functions will be exposed, or exported to applications that use your DLL. Sometimes you might have functions that are only useful to other functions in the DLL, so they might not need to be exported. The .EXP can be either somewhat complex or very simple, but either way it's just a text file. The Clarion 6 help file has lots of information on .EXP files, but it might be a little bit difficult to find. Just do a search for .EXP then look for Module Definition Files.

Naming your functions

As the world gets more and more crowded, and more and more functions are being written, and more and more third party products are being used, the possibility of naming collisions (functions, procedures, variables, equates from various developers being named the same thing) is becoming more and more prominent. I didn't put the `JS` at the beginning of the `MoveParentControl` function because I'm vain (okay, maybe I did). I did it because it is an attempt at preventing the possibility of using the same name by another developer. I guess someone else could use JS, but hey, I was here first! The same thing goes for API functions being prototyped. It doesn't at all hurt a vendor to put his/her prefix on that function, and it will save a lot of grief in the future.

Here's what the .EXP for our little DLL looks like:

```
NAME 'MYDLL' GUI
EXPORTS
  JSMOVEPARENTCONTROL@FlllUc       @?
Be sure to put at least one blank line here!
```

Notice the line that says `JSMOVEPARENTCONTROL@FlllUc`. You might notice that it doesn't look anything like the original prototype for `JSMoveParentControl()`. Why is this, you ask? I don't know. I don't think anyone knows. It just works. But what it really is, is a thing called Name Mangling. This is a fitting title, seeing how messed up the name appears. What name mangling does, is allow procedure overloading to be communicated when linking one entity to another. Name Mangling can also be turned off by using the `NAME()` attribute, and that might work okay for you for the most part, but it isn't a big chore to mangle the names yourself. Luckily, there's an example program under

%CWROOT%\Examples\src\pro2exp. Just compile that program and enter your prototype into the entryfield, then press the Tab key. Pro2EXP will show you your mangled prototype.

> **NOTE:** See that sentence about putting a blank line underneath the last export? That's very important, and was the cause of about an hour's worth of frustration on my part while writing this article. Before I added that line, the IDE (C5.5) kept crashing and would not export my function when it finally linked the DLL.

Compiling your DLL

Okay. Click on that lightning bolt and let 'er rip. Hopefully, like me, you get everything to compile correctly the first time. Ha ha. Okay, you finally got it to compile. Now what? After a successful compile of a DLL project, you end up with a .LIB file and a .DLL. The .LIB file is the glue that connects your app to your DLL. You place into the project of the app that makes use of the .DLL. When you connect the DLL to your application in this manner, you must have the DLL placed in a location where the application can find it. (See also, a trio of articles that Larry Sand wrote about Loading DLLs At Runtime.)

Sharing the DLL prototypes

I usually create an include file for including into my application so I don't have to do any typing, and all of my special functions are available to me. Simply copy the MAP section to a separate file. You'll use that in a moment.

Making your DLL available

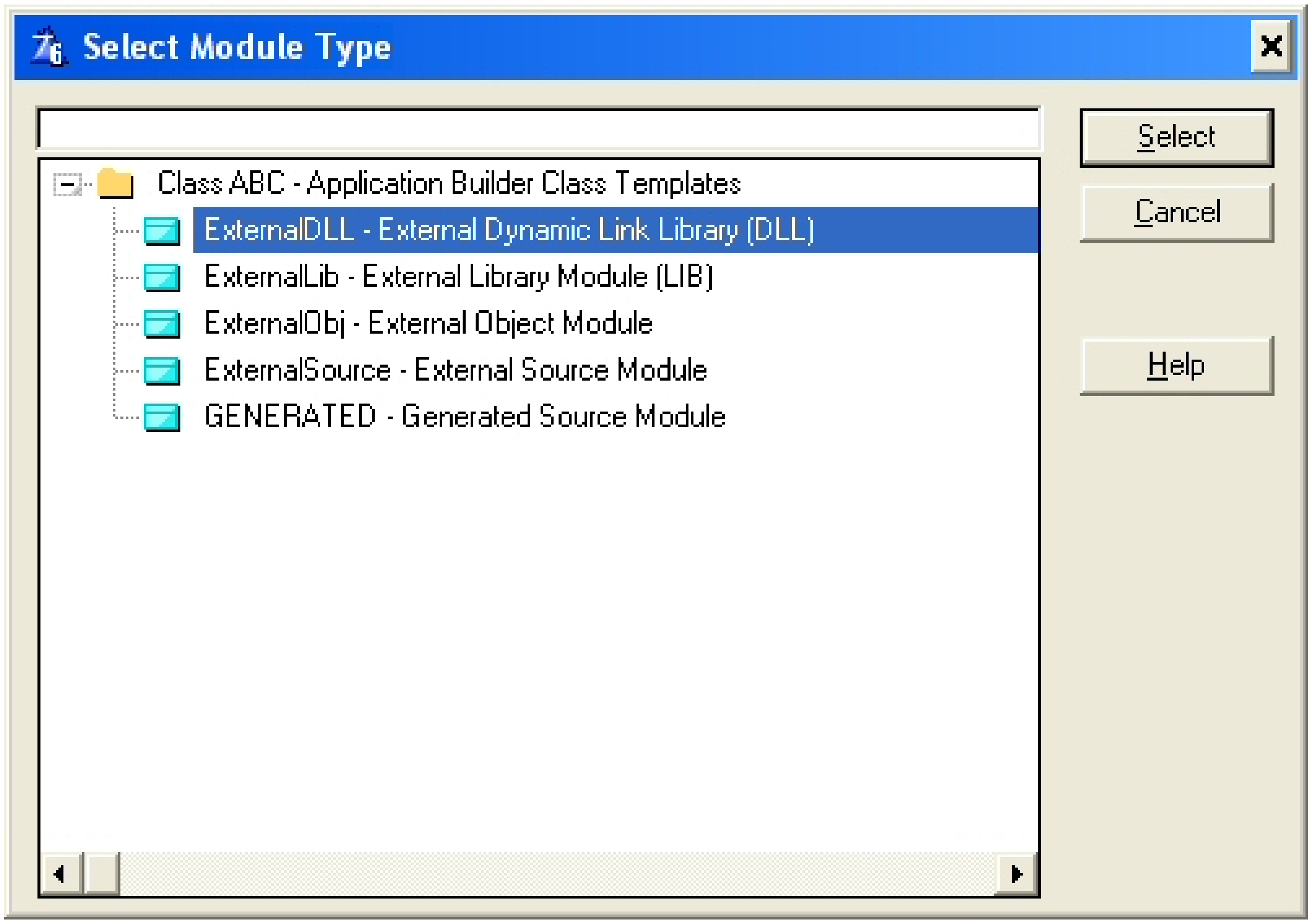
The Clarion environment has a few different folders for different files. The %CWRoot%\bin folder is where DLLs are usually placed. The %CWRoot%\lib folder is where the .LIB files are usually placed. What I usually do when I'm ready to publish my DLL (after testing) is run a batch file to clean up my DLL's folder and copy the LIB/DLL files to their proper places. Here is an example of what my batch file looks like:

```
copy mydll.dll e: \c55\bin
copy mydll.lib e:\c55\lib
copy mydll.inc e:\c55\libsrc
del *.bkp
del *.bak
del *.lnk
```

Using your DLL

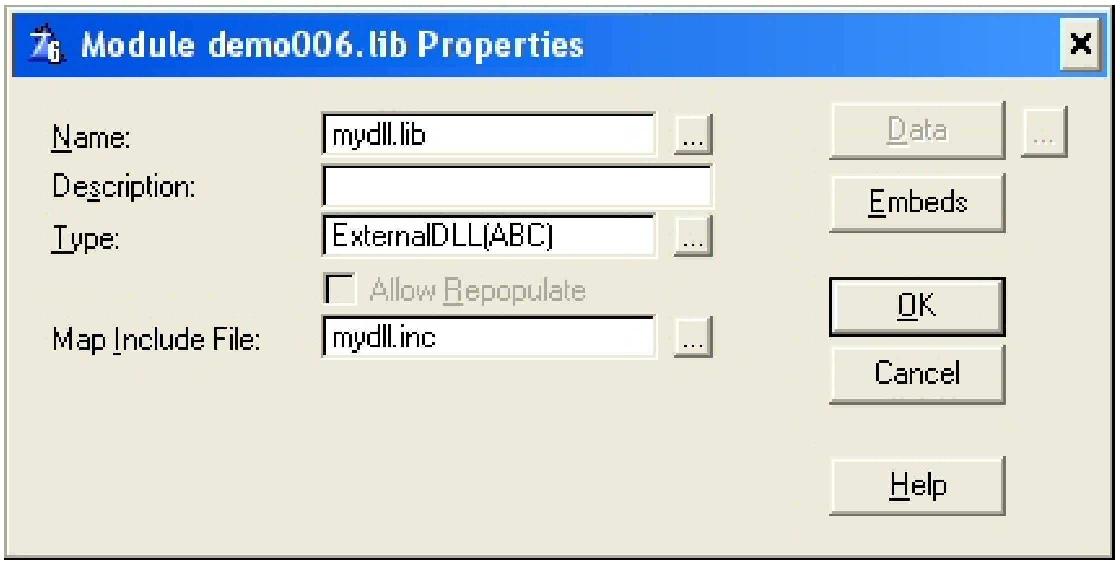
Now that you have your DLL, you're most likely going to want to make use of it. I guess if you wanted to be extra fancy, you could write a template that includes the DLL into your app. It's so simple to do, however, that I almost never bother with a template. Here's all that you have to do:

1. Open the App that you wish to add your DLL to.
2. Click on the Module tab.
3. Click on the Application menu item then select the Insert Module. A Select Module Type window will open.



Class ABC - Application Builder Class Templates
- ExternalDLL - External Dynamic Link Library (DLL)
- ExternalLib - External Library Module (LIB)
- ExternalObj - External Object Module
- ExternalSource - External Source Module
- GENERATED - Generated Source Module

4. Select External DLL. A module properties window will open.


Module demo006.lib Properties

Name: mydll.lib
Description:
Type: ExternalDLL(ABC)
Allow Repopulate
Map Include File: mydll.inc

5. In the Name field, enter the name of your DLL with a LIB extension.
6. In the Map Include File field, enter the name of your include file.

   NOTE: If you do choose to use an include file here, the procedures will not automatically be visible to the AppGen, although you can use them in embed points. It can be frustrating if you don't know about this behavior. If you need your DLL procedures to be visible to AppGen, you'll have to enter them manually instead of using the include file. Select the DLL's module, and press Insert or choose Procedure|New to create each procedure, making sure to set the procedure prototype as necessary. You may also want to check the Declare Globally option.

Now, if everything's set up correctly, you'll have full use of your DLL functions.

Looking back on this article, it might seem like a whole lot of hassle to go through for just a DLL, but it really is worth the effort. Once you get your special DLL set up, it's very easy to add new functions to it as time goes by, and you'll never have to code them again. If you do have to maintain that code again, it will only be in one place. And best of all, you won't have to fight with the AppGen over what declarations do or don't get exported. Isn't that nice?

Download the source from ClarionMag

Download this file from Jeff's ClarionFaq.com site

Carl Barnes wrote a utility template that makes use of the LinkName() function from the Clarion Template Language. CBMangle.zip contains the template, and CBSampleInputOutput.zip contains a sample input file and an example of what the output looks like. One big advantage to using the template language for generating an EXP is that the LinkName() function would most likely (hopefully) have the most correct logic in it for handling anything new in the language. Thanks, Carl.

Download CBMangle.zip

Download CBSampleInputOutput.zip

Download these files from Jeff's ClarionFaq.com site

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

Reader Comments

Add a comment

Clarion Magazine

Clarion Magazine's 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.

E-book format

All our e-books are unencrypted PDFs, and are formatted to the same standards as our print books. All you need to read them is Adobe Reader or any other PDF viewer.

Free e-books

If you take out a Clarion Magazine subscription or renewal by April 8, 2005, you'll automatically receive an electronic coupon for **a free e-book**! You can verify your coupon by logging in to your My ClarionMag page.

Free updates

We're constantly updating our e-books with the latest articles. All e-book purchases come with a three month free upgrade policy; if you're also a subscriber, you get unlimited free updates, as long as you have an active subscription.

Subscriber discounts

Clarion Magazine subscribers can purchase e-books at discounted prices.

Current and upcoming e-books

E-Book	Your Price	Regular Price	All prices in US Dollars
E-Book (PDF): Clarion Edit-In-Place Tips & Techniques Clarion's Edit-In-Place (EIP) capability is powerful, but difficult to master. This extensive e-book covers not just the standard EIP techniques, but also some very cool tricks with forms in place of EIP, and checkboxes for managing many-to-many relationships. **Regular price $9.95/19.95** View the table of contents	**$7.95**	$16.95	Add to Cart
E-Book (PDF): Learning The Clarion Template Language The real power of Clarion is its template-based code generation. Just like the shipping templates, your own custom templates increase your productivity and reduce the effort required to maintain code. And writing templates is easier than you think, as this intro-level ebook shows. Topics include template language basics, code templates, extension templates, reusable template #GROUPs, and the Template Wizatron/Writer. View the table of contents	**$9.95**	$19.95	Add to Cart

E-Book (PDF): Mastering Clarion DLLs (version 1.1) Almost any Clarion application can benefit from being split into an EXE and one or more Dynamic Link Libraries (DLLs). This collection of articles shows all the tricks for getting the most out of DLLs, from how to easily split your applications up for easier maintenance, to calling unlinked DLLs at runtime, to rebasing your DLLs (and third party DLLs) for greatly improved load times. An essential handbook for anyone who develops large applications. View the table of contents	**$9.95**	$19.95	Add to Cart
E-Book (PDF): Learning The Clarion Language Many Clarion developers begin writing applications with the AppGen, and then find themselves wanting to do more with the Clarion environment. But learning how to write Clarion code by examining the generated code can be overwhelming. This ebook begins with an overview of the Clarion environment, and by using simple examples shows how easy it is to write Clarion code. Topics include standard Clarion data types and equates, creating procedures, list box formatting, and basic file handling techniques. View the table of contents	**$9.95**	$19.95	Add to Cart
E-Book (PDF): Threading In Clarion Clarion 6 has opened up a new, exciting, and potentically confusing world of threading to Clarion developers. This collection of articles will guide you through the various new functions, classes, and techniques available to unleash the power of true threads in your applications. As a bonus, you also get Jim Kane's classic articles on using API threads with Clarion 5.x. View the table of contents	**$9.95**	$19.95	Add to Cart
E-Book (PDF): More E-Books Coming in April/May We have a number of additional e-books on the way! Upcoming topics include debugging, COM, Edit-In-Place, using C/C++ code, and ABC design notes - stay tuned!	**$9.95**	$19.95	Available April, May 2005