Clarion Magazine

Clarion News

- » Combit's Easter Egg Hunt
- » vuWMI Demo
- » Free WMI Template
- » vuMail 3.13
- » DMC 2.0.0.1 Spring Special
- » Capesoft Profiler Gold Release
- » GPF Reporter Supports C7.1
- » CHT's Free C7 Batch Compile Generator 1.06
- » vuMail 3.12
- » Helderberg Clarion User Group Meeting
- » vuMail Beta 3.12
- » DMC BugFix
- » Codejock Wrapper Templates Updated
- » Clarion 7.1.6849
- » PropertyGrid 1.13
- » DMC 2.0.0.0
- » Free Batch Compiler For C7.1
- » CalendarPro 2.03
- » vuMail 3.11
- » vuFileTools 3.51
- » Noyantis Installer C7.1 Compatibility
- » amazingGUI 1.2.1.2
- » SetupBuilder 7.2 Build 2884
- » Query Wizard 7.03
- » HTMLEditor Wrapper 1.01
- » Icetips Window Fixer
- » ReportControl Wrapper 1.13
- » Blog: Setting Previewer Output Targets

Latest Subscriber Content

Advertising Feature: A Journey Towards An Employee Portal Using Clarion.Net

CCS Technologies is a Software Development and Services Company established in 1980, providing Clarion outsourcing services.

Posted Monday, March 08, 2010

Did You Get An Email From Us?

On Friday we sent out a mailing list reminder email. Did you get it? If not, here's what you can do.

Posted Monday, March 08, 2010

Converting C6 Apps to C7, Part 1

Clarion 7 is without doubt one of the largest changes in the history of Clarion, thanks to its completely new IDE and a reworked code generation system that is a lot stricter about APP and DCT data. As a result, you're likely to encounter some errors when converting your APPs to C7.x. Philip Prohm takes a detailed look at the common problems and their solutions. Part 1 of 2.

Posted Wednesday, March 10, 2010

Converting C6 Apps to C7, Part 2

Clarion 7 is without doubt one of the largest changes in the history of Clarion, thanks to its completely new IDE and a reworked code generation system that is a lot stricter about APP and DCT data. As a result, you're likely to encounter some errors when converting your APPs to C7.x. Philip Prohm takes a detailed look at the common problems and their solutions. Part 2 of 2.

Posted Thursday, March 11, 2010

Source Code Library 2010.02.28 Available

The Clarion Magazine Source Code Library has been updated to include the latest source. Source code subscribers can download the February 2010 update from the My ClarionMag page. If you're on Vista or Windows 7 please run Lindersoft's Clarion detection patch first.

Posted Thursday, March 11, 2010

New Newsgroup Server Online

We've switched to our new news (NNTP) server, and you should have access again as soon as the DNS change reaches you. If you have difficulty, try unscribing from (and perhaps deleting) the newsgroups and resubscribe. And if you haven't yet joined the ClarionMag newsgroups, please do! See the My ClarionMag page for login details.

Posted Thursday, March 11, 2010

Where Delete Occurs, Part 1

Life, opines Dr. Parker, was so much easier before ABC and Edit-in-Place and allowing users to delete records directly

- » Noyantis Acquires LGP, gCalc, gFileFind
- » iQ-XML 2.60
- » gCalc 5.1 Updated for Clarion 7.x
- » TraceIt Updated
- » SetupBuilder 7.1
- » Super Tagging 7.01
- » EasyListView 1.03
- » Super Security 7.00
- » ProScan and ProImage For Clarion 7.1
- » RPM for 7.1.6695
- » Icetips February Newsletter

[More news]

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

[More Clarion & .NET]

[More Clarion 101]

Latest Free Content

- » Advertising Feature: A Journey Towards An Employee Portal Using Clarion.Net
- » Source Code Library 2010.02.28 Available
- » New Newsgroup Server Online

[More free articles]

Clarion Sites

Clarion Blogs

from the browse. So how can you intercept delete operations if you want to add some specialized record handling? Part 1 of, well, a bunch.

Posted Tuesday, March 23, 2010

Where Delete Occurs, Part 2

Steve Parker continues his journey into the bowels of ABC record deletion, and answers the question of where, exactly, delete does occur.

Posted Tuesday, March 23, 2010

Where Delete Occurs, Part 3

Ever since CPD, Clarion developers have enjoyed "set it and forget" relational integrity. But there are times, as Steve Parker knows, when you can do a better job with your own RI code.

Posted Monday, March 29, 2010

Where ... Stuff Occurs: Deriving the FileManager

Sometimes embedding custom CRUD code in browses or forms is the right way to go, but if you want to apply that code application wide, you might consider deriving the FileManager. Steve Parker examines the pros and cons of this approach.

Posted Tuesday, March 30, 2010

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

Source Code

The ClarionMag Source Code Library

Clarion Magazine is more than just a great place to learn about Clarion development techniques, it's also home to a massive collection of Clarion source code. Clarion subscribers already know this, but now we've made it easier for subscribers and non-subscribers alike to find the code they need.

The Clarion Magazine Source Library is a single point download of all article source code, complete with an article cross-reference.

More info • Subscribe now

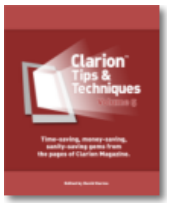
Printed Books & E-Books

E-Books

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

Printed Books

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



- » Clarion Tips & Techniques Volume 5 - ISBN 978-0-9784034-1-6
- » Clarion Tips & Techniques Volume 4 - ISBN 978-0-9784034-0-9
- » Clarion Tips & Techniques Volume 3 - ISBN: 0-9689553-9-8
- » Clarion 6 Tips & Techniques Volume 1 - ISBN: 0-9689553-8-X
- » Clarion 5.x Tips and Techniques, Volume 1 - ISBN: 0-9689553-5-5
- » Clarion 5.x Tips and Techniques, Volume 2 - ISBN: 0-9689553-6-3
- » Clarion Databases & SQL - ISBN: 0-9689553-3-9

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

From The Publisher

About Clarion Magazine

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

Subscriptions

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

Satisfaction Guaranteed

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

Dave Harms

ISSN

Clarion Magazine's ISSN

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

About ISSN

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

Clarion Magazine

Clarion News

Search the news archive

Clarion.NET Training

Gus Creces is offering an on-line Clarion.NET Training Seminar series, based on a free digital book written by Charles Petzold. Gus will use .NET Book Zero as a bridge from the C-Sharp (C#) language into Clarion-Sharp (Clarion#).
Posted Thursday, April 08, 2010

CHT Free C7 Batch Compile Generator 1.07

CHT's Free C7 Batch Compile Generator version 1.07 now posted for download. In this release the expiration limitation has been removed.
Posted Thursday, April 08, 2010

CHT 50% Off Subscription Drive

CHT is offering a "Target 100" offer at a discount of 50%. The 2010 "Target 100" order page will stay up until there have been 100 new subscribers who purchase this offer. At the end of the day on which the "Target 100" objective is reached (or exceeded) the page will come down.
Posted Thursday, April 08, 2010

Clarion2Java On SourceForge

Clarion2Java is now a SourceForge project. The compiler and runtime are being used for a mid size application (350+ forms/browses/reports/etc) ported from Clarion 5.5, but the project it is still very young. The current file driver is heavily tuned for PostgreSQL. The project code is licensed under the Lesser LGPL, which means that you can use it, ship it and distribute without constraint. But if you make modifications to clarion2java itself - i.e. bug fixes to the compiler, those modifications must also be under an LGPL licence. Other licencing can be arranged on request if necessary.
Posted Thursday, April 08, 2010

PropertyGrid Wrapper 1.16

Version 1.16 of the PropertyGrid Wrapper template is available. Modifications include: Multi Threading enhanced; Category / Item ID variable increased; 'InPlace Buttons' added to Item definition; New method added - AddItemInPlaceButton; New method added - GetInPlaceButtonProperty; New method added - SetInPlaceButtonProperty. The new version can be downloaded from the Members area using the original download and registration details contained in your sales email.
Posted Thursday, April 08, 2010

View Wizard Version 7.02

ClarionTools is has released View Wizard version 7.02. This release is free to all registered View Wizard 7 developers. Existing customers are encouraged to update to this latest version. View Wizard supports ABC and Legacy Template Chains, for Clarion C55, C61, C62, C63, C70, and the latest C71 release. Changes include: Alternating Row Color (Greenbar) support with simple global activation; Added vwSortFieldMatch(STRING,STRING) function that is useful to determine if a locate is occuring on a specific field to further format output for the resulting locate; Window Resize extension template; Can now select where the column headings are displayed from; Column heading can be applied at runtime from the column heading in the dictionary or field definition; Can selectively turn off the dictionary options if desired; Picture format of the browse field can be applied at runtime from an override, or variable; Column freezing support that works mid-column; Can modify the alternate column description with static or runtime variable.
Posted Thursday, April 08, 2010

Combit's Easter Egg Hunt

Around this year's Easter holidays, more precisely from March 29th to April 11th, customers can hunt for Easter eggs with different discounts on the website of German software manufacturer combit. The sales promotion applies to the regular list price of development component List & Label and CRM solution combit Relationship Manager. "As in the real world, the most wanted Easter eggs have been hidden thoroughly," explains CEO Christiane Koerdel. "It's worth looking closely at every corner of our website."
Posted Wednesday, March 31, 2010

vuWMI Demo

A demo of vuWMI, compiled in Clarion 7.1, is now available. There are approximately 450 items on the dropdown list.
Posted Monday, March 22, 2010

Free WMI Template

Bill Roe has released a free, unsupported template with a couple of wrapper functions for Windows' WMI Object. These functions work fine in Vista and Windows 7 but may not be reliable in Windows XP SP2.
Posted Monday, March 22, 2010

vuMail 3.13

An update to vuMail is available for download. This update incorporates a new feature to aid in embedding multiple images into the body of an email. The vuMail Help File has also been updated to include some better information on sending emails.
Posted Monday, March 22, 2010

DMC 2.0.0.1 Spring Special

Until April 3 2010 you can get DMC at a 30% discount. Purchase a new Enterprise licence for $190 and save $80 (estimate of prices in US Dollars depending on bank exchange rates). Renew your Enterprise licence for $56 and save $24. You will require a "Spring Special" coupon code - please use *DMC-45K2D462JH* and get your immediate discount applied.
Posted Monday, March 22, 2010

Capesoft Profiler Gold Release

Capesoft Profiler has gone gold, and the $100 discount off the regular gold price ends March 26, 2010. The beta price was $299, but 'til Friday it'll be $247 - and on Saturday it'll go up to the regular gold price of $347.

Posted Monday, March 22, 2010

GPF Reporter Supports C7.1

The code necessary to support Clarion 7.1 has now been added to GPF Reporter.

Posted Monday, March 22, 2010

CHT's Free C7 Batch Compile Generator 1.06

CHT's Free C7 Batch Compile Generator version 1.06 is now available. Changes include: A manifest that displays the version number when the cursor is hovered over the .EXE from windows File Manager; Shows the CHT application icon when the .EXE is displayed from Windows File Manager; "Solutions Found" popup window now asks if you would like to run COMPILE.BAT immediately; EXE oes not need to be copied to the solutions directory and may be placed in a pathed directory from where it will redirect correctly into your solutions folder; Fix for version 1.05 bug where the Clarion.NET path could be returned for ClarionCL.EXE if both Clarion.7 and Clarion.NET were installed on the same machine

Posted Monday, March 22, 2010

vuMail 3.12

vuMail beta version 3.12 is available for download. This version corrects the case sensitivity for embedding images and includes two new functions: vuSetCodePage() and vuGetCodePage().

Posted Monday, March 22, 2010

Helderberg Clarion User Group Meeting

The next HCUG meeting will be held 17th March 2010 (Wednesday) from 17:30 for 18:00. Duration: around 2-3 hours. Venue: Unit 304 Oakmont Building, Somerset Links Office Park (Off De Beers Ave), Somerset West. Bring: Yourself and a colleague, Oh .. also cool drink or beer, of your preference / choice. Cover charge: R40 per person cash (for meal, snacks, soft drinks, coffee, tea etc). Topics at meeting: Migrating from 6.X to 7.1 (by Ben Dell); Clarion 7.1 Demonstration (by Ben Dell); More on TPS to MSSQL, on request (by David Swindon); Q + A.

Posted Monday, March 15, 2010

vuMail Beta 3.12

vuMail beta 3.12 is available for download. This version corrects the case sensitivity for embedding images and includes two new functions: vuSetCodePage() and vuGetCodePage().

Posted Monday, March 08, 2010

DMC BugFix

A bugfix to the DMC gold release is available. This fixes problems with registration and maintenance plan renewals.

Posted Monday, March 08, 2010

Codejock Wrapper Templates Updated

New versions of all of the Codejock Wrapper templates are now available. These now support the recently released v13.3.0. Also, all of the Classes now include two new methods: GetProperty and SetProperty. These make it easier for you to read and/or set ActiveX Control properties that might not be covered elsewhere within the class. The new versions can be downloaded from the Members area using the original download and registration details contained in your sales emails.
Posted Monday, March 08, 2010

Clarion 7.1.6849

SoftVelocity has released Clarion 7.1 build 6849.
Posted Monday, March 08, 2010

PropertyGrid 1.13

Version 1.13 of the PropertyGrid Wrapper template is available: New Child of Child facility added; New method added - GetCategoryExpanded; New method added - GetCategoryProperty; New method added - GetItemExpanded; New method added - GetItemProperty; New method added - GetProperty; New method added - SetCategoryProperty; New method added - SetItemExpanded; New method added - SetItemProperty; New method added - SetProperty; Codejock v13.2.2 compatibility added. A mock up example of the new Hot Grid facility has been added. The plan is to allow the developer to attach the PropertyGrid to an existing Clarion browse and the various Categories and Items will be populated from the Browse contents much the same as the ReportControl Browse Enhancer works. The contents of the Hot Grid will then dynamically change when a new record is selected within the Browse. More news on this including an estimated release date will be available shortly. The new version can be downloaded from the Members area using the original download and registration details contained in your sales email.
Posted Monday, March 08, 2010

DMC 2.0.0.0

DMC Gold 2.0.0.0 is now available. If your Maintenance Plan is expired you will need to renew your licence. The registration method has been modified in this version and you will need to register your application after applying this new version. A lifetime license is also now available. Features include: Backup an entire SQL Data Base in a single Task; Clone an entire SQL database (multiple Tables selection) to any existing or new other SQL database in a single Task; Clone a folder of DAT or TPS files; Transfer a folder of DAT or TPS files to SQL; Do SQL lookups during transfers; New version of DMC runtime engine for distribution.
Posted Monday, March 08, 2010

Free Batch Compiler For C7.1

If you've got the latest Clarion 7 (C7.1.0.6813 or later), you may find this free utility useful for bulk compile of single app solutions. This is the utility used to batch compile the CHT C7 demo applications.
Posted Monday, March 01, 2010

CalendarPro 2.03

Version 2.03 of the CalendarPro Wrapper template is available, along with an updated demo. Modifications include: 'Allow All Day Events to be Moved' option added; 'Allow All Day Events to be Resized' option added; 'Allow Events to be Moved' option expanded; 'Allow Events to be Resized' option expanded; 'Allow Recur. Events to be Moved' option expanded; 'Allow Recur. Events to be Resized' option expanded; New Calendar method: 'AllowAllDayEventMove';

New Calendar method: 'AllowAllDayEventResize'; New DatePicker method: 'MoveSelection'; BUG FIX: DeleteAllEvents now clears memory Cache; BUG FIX: Custom Colours could disappear when double click used and not reappear immediately afterwards; BUG FIX: 'Event Moved' embed point not firing under certain circumstances within multiple Schedules; BUG FIX: GetSelected...Date() could return incorrect value in Week Summary and Month view. The new version can be downloaded from the Members area using the original download and registration details contained in your sales email.
Posted Monday, March 01, 2010

vuMail 3.11

vuMail 3.11 is a free interim BETA release that includes the ability to automatically import and embed images in the body of an email. See "What's New" in the help file for a complete explanation. No changes to any source code is necessary to use vuMail 3.11 (just replace the vuMail.dll).
Posted Monday, March 01, 2010

vuFileTools 3.51

A new maintenance release of vuFileTools is available for immediate download. vuFileTools ver 3.51 is a free update. Changes include: vuRecordWav, spaces are now allowed in the name and path; vuISOS64(), function wasn't exported; Updated memory calls to work on Windows 98; vuCopy() the new updated allows you to copy wildcard files in both the root and subdirectories, but at the cost of speed and the loss of being able to use a UNC for the path, vuCopy() now allows you to choose which method to use; vuFindFile(Filename, Location) can locate a file and return the complete path and name (restricted to where you want to search, can have multiple location criteria).
Posted Monday, March 01, 2010

Noyantis Installer C7.1 Compatibility

All of the Noyantis template installers are now C7.1 compatible. Please fully uninstall any previous versions prior to using the new installers.
Posted Monday, March 01, 2010

amazingGUI 1.2.1.2

amazingGUI 1.2.1.2 has been released. Users with an active suscription will recieve installation keys by mail. Changes include: In multi-DLL projects the data application was trying to export the RunTime Enabling variable even when RunTime Enabling function was not selected; There was a missing function when compiling LOCAL.
Posted Monday, March 01, 2010

SetupBuilder 7.2 Build 2884

A pre-release version of SetupBuilder 7.1, Build 2884 is now available. If you would like to get access to SetupBuilder "Development Builds" (pre-releases), please send your serial number to sales at lindersoft.com. A current maintenance and support plan subscription is required.
Posted Monday, March 01, 2010

Query Wizard 7.03

ClarionTools has released Query Wizard Version 7.03. This release is available immediately and is FREE to all

registered Query Wizard 7 developers. Existing customers are encouraged to update to this latest version. Query Wizard supports ABC and Legacy Template Chains, for Clarion C55, C61, C62, C63, C70, and the latest C71 release.

Posted Monday, March 01, 2010

Clarion Magazine

Advertising Feature: A Journey Towards An Employee Portal Using Clarion.Net

by Thomas Mathew (tmathew@ccstechnologies.in)

PAID ADVERTISEMENT

CCS & Clarion.Net Development

At CCS Technologies we have been proving our expertise in Clarion.Net right from the initial beta version, our first endeavor being 'My Portal', a feature-rich Employee Portal.

My Portal is the gateway that unifies access to all information and applications among employees and management in a company. It is a tool that helps a company manage its data, applications, and information more easily, and through personalized views.

MyPortal helps to:

- Publish notices and make announcements to the employees.
- Allow employees to share their information and knowledge by posting articles and white papers.
- Make the company environment more synergistic by sparking off discussions between employees including technical and non-technical topics.
- Help the employees to organize their work by facilitating them with task scheduling options.
- Provide the employees a market space where they can advertise to sell and buy their personal goods.
- Help the employees to get the first-hand information (both personal and official) of their colleagues.
- Provide an album space for the employees so that they can upload pictures and view others.

- Conduct surveys quickly and effectively and collect feedback regarding various company matters.
- Switch the work-flow based company activities like Leave application, Expense Reimbursement, Material Request etc. from paper to the portal.

For better understanding, one of the work-flow based activity is explained below.

Leave Application Form

Leave Application flows from the employee to the Dept Head (or approver), who can then approve or reject it. On approval, the form will flow to the next approver (normally HR Dept). A mail is sent to the employee informing him of the status.

CCS & Clarion

CCS Technologies is a Software Development and Services Company established in 1980. Headquartered at Cochin, India we have presence in USA, UK, Australia and Middle East. We started using Clarion in 1992, with CPD and now in C6.3, C7.1 & Clarion.NET. We provide following Clarion Services on an Offshore Delivery Model to our overseas clients.

Clarion Offshore Services

- Developments in Clarion 6.x and 7.1
- Conversion from Clarion 5.x, 6.x to 7.1
- Clarion.Net — Applications to run on Desktop, Web and Mobile devices
- Maintenance in Clarion Applications — (New Functionality, Reports and Bug fixing)
- Database Migration to MSSQL/Oracle/MySQL/Sybase SQL
- Development of New Templates, Maintenance in Legacy/Third-party Templates
- Clarion Integration with other applications developed in Java/PHP/.Net using Web services and SOAP protocol

Benefits in outsourcing

- Reduced Development Cost
- Resource Scalability, as and when needed
- Leveraging Time Zone Advantages

- Reduced administration of Human Resources

Outsource Management model

- Project Management by Partner or CCS
- CCS Developers working as extended team members of the Partner
- Directly reporting to the Partner Project Lead
- Weekly Time Sheets
- Progress review meetings by both parties

For more details, please write to us at busdev@ccstechnologies.in

- ***Reliable***
- ***Innovative***
- ***Cost effective***
- ***On-time***
- ***Friendly***

www.ccstechnologies.in

PAID ADVERTISEMENT

Clarion Magazine

Did You Get An Email From Us?

Published 2010-03-08

Did you get an email from Clarion Magazine?

On Friday we sent out reminder emails to everyone on the ClarionMag mailing list. If you are on that list, and you didn't get an email, it could be that:

- We don't have your current email address

or

- The email was incorrectly blocked/removed by a spam filter, either on your mail server or on your own computer

If you didn't receive an email from us with the subject "Clarion Magazine mailing list reminder", sent by subs at clarionmagazine.com, then there are a couple of things you can check.

- If you're not registered at ClarionMag, you can do so here: http://www.clarionmag.com/cmag/newmember.frm
- If you are already registered, make sure we have your current email address by logging in to http://www.clarionmag.com/cmag/userupdate.frm; while you're there make sure you're on one or more of our mailing lists. We use a double opt-in mailing list, so after you register for a mailing list you'll receive an email with a confirmation link. Make sure you click on it.
- Add our email addresses to your address book and/or your email program's whitelist.
- If you have access to your mail server, consider whitelisting the clarionmagazine.com domain.

If you have any difficulties please contact us.

Reader Comments

Add a comment

Clarion Magazine

Converting C6 Apps to C7, Part 1

by Philip Prohm

Published 2010-03-10

Clarion 7 is without doubt one of the largest changes in the history of Clarion. The nearest rivals would be the transitions from Clarion for DOS to Clarion for Windows and from the Clarion (a.k.a. Legacy) template set to the Application Builder Class (ABC) template set. All three, for different reasons, required fundamental changes to the way we were used to doing things. This time around the paradigm shift is the totally new Integrated Development Environment (IDE) – essentially we now have a Microsoft Visual Studio-style IDE (although it is more correct to say we have an IDE based on code licensed from the creators of an open source alternative to Visual Studio).

The new IDE (based on .NET Framework 2.0 and higher) includes – among other things – a new dictionary editor, a new application generator (now 32-bit), a new build system (based on MSBuild), new dictionary and application text file formats, a new container file which can contain multiple applications (called a Solution and based on the Visual Studio approach) and a new capability to have multiple applications open simultaneously. There are a few changes to the Clarion language and the template language as well.

Not surprisingly, the potential incompatibilities of Clarion 7 with Clarion 6 are many. SoftVelocity has made a committed effort to minimise the work required by you to convert Clarion 6 applications to Clarion 7 and, to their credit, have been largely successful in this endeavour. In this article I will guide you through the conversion process and present some solutions to problems that can arise.

Getting Started

Copy your Clarion 6 ("V6") directory tree in its entirety and rename one of the two trees such that it is clearly labeled as V6. For example, if you have a directory tree rooted at d:\src\widget you will finish with two identical trees named

d:\src\widget.cw6

and

d:\src\widget

This way, there can be no chance of you unintentionally changing something in the old V6 tree while still having the old tree to refer to on occasion. You can (and should) additionally make a backup of the V6 tree using your conventional backup regime.

Use d:\src\widget for your V7 work (for one thing, to prevent problems that can result from using different directory paths compared to your V6 project) and take the opportunity to prune the V7 tree (d:\src\widget) of any accumulated cruft

you know you won't need or consciously decide you are going to delete anyway. If in doubt regarding any particular file, keep it for now and decide later when you're finished converting.

Start Clarion 7 ("V7").

On the Start Page, click on Open Solution button, browse to d:\src\widget and Open widget.app.

You'll be prompted as to whether you want to irreversibly convert your APP file. If you have copied your directory tree per above, click OK to continue – your V6 APP file will be renamed and not deleted.

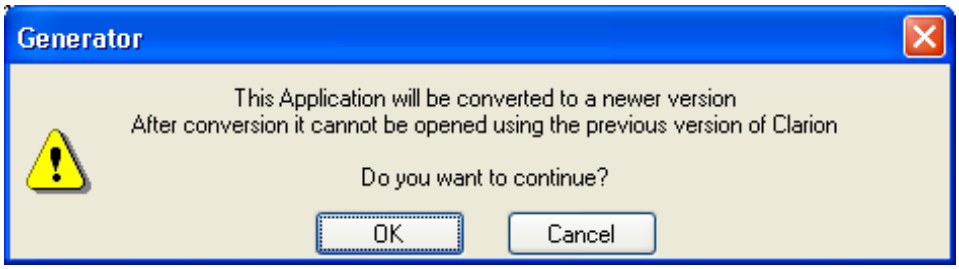


Figure 1. Converting an APP

You'll also be prompted as to whether you want to irreversibly convert your DCT file. Again, if you have copied your directory tree per above, click OK to continue – your V6 DCT file will be renamed and not deleted.

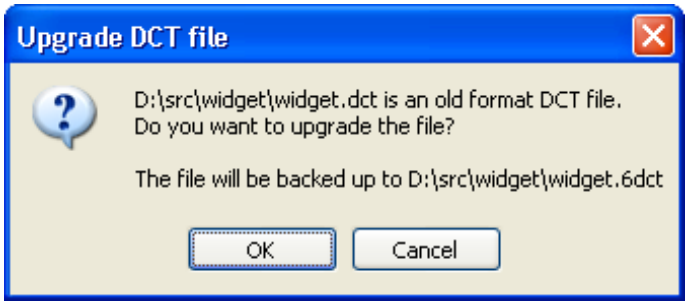


Figure 2. Upgrading a DCT

If C7 needs to make any changes to a dictionary that was opened in C6, it will write out a change log with the name <dictionaryname>.Changes.log. In my example the name of that file is d:\src\widget\widget.Changes.log.

Note: If you happen to have open the change log file from a previous run, you may need to close it before proceeding: if the program you used to open the log file opened it with exclusive access, V7 will not be able to write a new one.

Figure 3. Dictionary changes logged

While AppGen is converting your APP to V7, it might encounter some errors in some WINDOW or REPORT structures. I'll cover this scenario in Part 2.

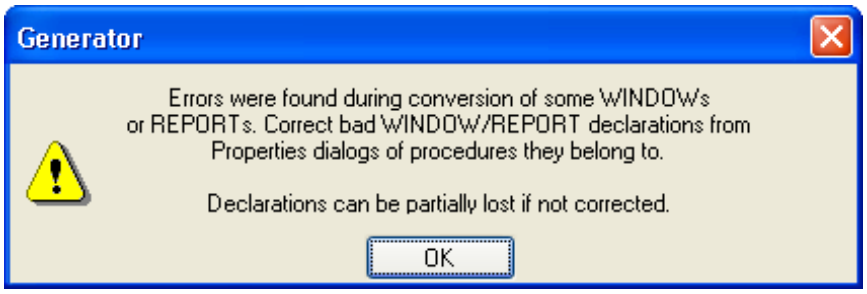


Figure 4. Conversion errors

At this point, the new APP file will be loaded. If the first APP you attempt to convert to V7 is a data DLL then the V7 APP will, in most cases, load without incident.

However, if your APP uses any templates that happen not to be registered in the V7 IDE, now is when you'll discover the fact. This latter scenario will be covered later in Unregistered Templates but for the moment let's deal with the former scenario where there are no unregistered templates.

Press Ctrl+M (Build|Generate and Make Current Application) or click on the Generate and Make Current Application toolbar icon. If the Output window (Ctrl+Alt+O) displays "Build Finished Successfully", you're done for this APP.

Figure 5. Generate and Make

Screen dumps for this article were taken from Clarion 7.1.6545. Other versions do differ in cosmetic detail.

That was easy, wasn't it? Now let's deal with some real-world problems. First the dictionary.

Dictionary Problems

Have a closer look at the dictionary changes log file, which you can view in the IDE via View | Data Dictionary | Data Changes). It's tab-delimited so you can load it into a spreadsheet rather than a text editor if you wish. Using a spreadsheet allows you to sort the columns, which have headers (Name, Type, Change, Cause).

Here is a short example of a changes log file:

```
Name          Type     Change Cause
FIL:SysID     Field    Derived From attribute removed
                       POO:ID is a different type
TBL:ProdCode           Derived From removed
                       DNE:DNECode was not found in the dictionary
```

The first line of the file has four column headers and on subsequent lines it has information for each change. The Type column is sometimes blank. The Name column is the name of the dictionary column that V7 has had a problem with and the Change column is the action V7 has taken as a workaround in order to load your dictionary and application. Because

the file column widths can vary significantly, the file will look cleaner if loaded into a spreadsheet program compared to a text editor.

In a spreadsheet, sort by "Change" then "Cause". Some possible Changes are:

- Derived From attribute removed
- Derived From removed
- External Name changed to <new_name>
- Label changed to <new_label>
- Freeze Set

The first two are essentially the same but arise from different causes. The main causes of these two changes are that after a second field was derived from the first, the type of the first field was changed, the name of the first field was changed, or the prefix of the file containing the first field was changed.

All of these are fairly trivial and won't stop your converted APP compiling but they do have the consequence that the derived field is orphaned from the derived-from field, and if you depend on derived fields inheriting changes made to the parent field then you will want to restore the relationship. You can do this two ways.

One is by editing the original V6 dictionary, changing the parent field or file to the state it was in when the second field was derived from it, and importing the edited V6 dictionary into V7 a second time.

The other way is by simply going through the list of fields and deriving the child field from the parent again. But *be warned* that the child field will be changed to match the parent field. For example, if the parent field is a string and is now longer than the child field, the length of the child field will be increased. This may not be what you want. In all, it may be simpler to accept the break in connection between the derived field and the one from which it was derived, and deal with them on a case-by-case basis later after finishing the rest of the port.

I believe the reserved word setting is designed for forward compatibility with .NET. And in most cases I suspect you can safely turn it off.

File, field and key labels are changed if they conflict with reserved words. In earlier versions of Clarion you could get away with this in certain circumstances. The external name is set to the old label so you'll still be able to access your files/tables as before, as far as the generated code goes. When you eventually generate and compile, the compiler will catch occurrences of the old label in your embed points (which you can fix at that time) but you should also search all source for occurrences of the new label to ensure no code has been broken. Another approach is to run a differences program and compare your C6 generated code with your C7 generated code (which you may want to do in any case).

Comparing source is a topic close to my heart but unfortunately beyond the scope of this article. Suffice to say that a differences program is one which will output the differences between two files or two sets of files in text or graphical format. A free example of the former is fc which is part of Windows and you can run from the command prompt. Enter fc /? for online help. A free example of the latter is WinDiff which has been included as part of the Windows CD since Windows 2000 – run setup.exe from within \support\tools.

If you've set up your redirection file to place your sources in a dedicated directory you can simply run differences on your two source directories (eg. d:\src\widget.cw6\~src and d:\src\widget\~src). Out of the box, you can can run differences on the two application directories (d:\src\widget.cw6 and d:\src\widget) and ignore binary files (which are almost certain to be different). If your differences program allows you to specify a filter, for example only comparing *.clw and *.inc, so

much the better. Other file extensions such as .equ, .trn and .int represent source files but typically they are not generated by AppGen, unless you have a third party template that generates them, so they will not need to be included in the comparison.

In summary, you need to assess the changes listed in the changes log file in terms of the impact they can have on your application. However, unless you want to edit the V6 DCT and import it a second time you may find you decide you don't have to make any changes to the V7 DCT until after you have generated source.

Unregistered Templates

Before your APP will load successfully you will need to register all the templates the V6 APP used. What about the unregistered ones? Some of your required templates you could guess easily and simply go ahead and register them.

Close any open APPs, but don't save them if you have template registration errors (as explained below).

Choose Tools|Edit Template Registry menu command. Click on Register button, browse to one of C:\Program Files\SoftVelocity\Clarion71\template\win or C:\Program Files\SoftVelocity\Clarion71\accessory\template\win (or wherever your desired template file resides) and Open the desired template file.

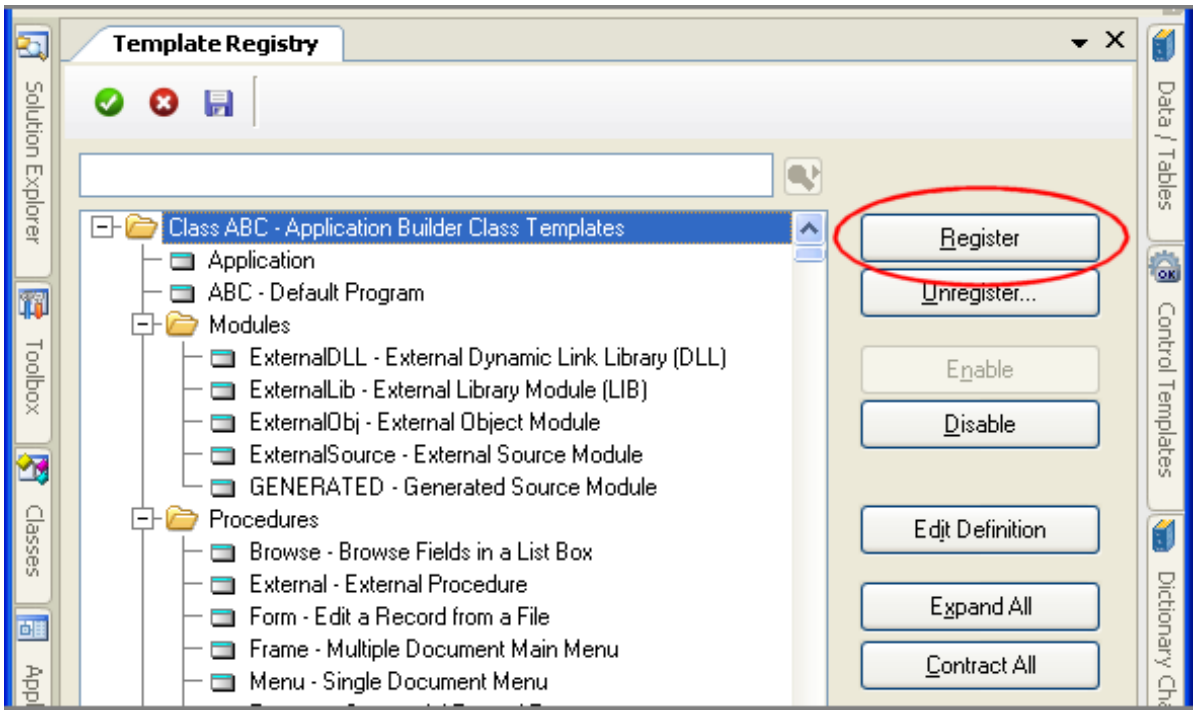


Figure 6. Register a template

Compatible templates will appear in the template tree without incident; any incompatibilities will need to be edited by either you or the vendor of the template before retrying this step. Most vendors have now updated their templates, and most errors are quite easy to fix, such as syntax errors which C6 allows by C7 doesn't. The most common of these is a missing END statement after a CONTROL group. Russ Eggen has also written about some more complex conversion issues with control templates.

When finished registering templates, click on the Save and Exit AppGen toolbar icon (a white tick in a green circle) or choose File|Close|File menu command. (The keyboard shortcut Ctrl+F4 appears not to work.) If you use the menu command and any changes to the Template Registry have not yet been saved, you will be prompted as to whether to save or discard those unsaved changes or to simply Cancel.

Discovering required but unregistered templates

Perhaps you've forgotten which templates your APP needs and which ones it doesn't. After all, you installed and registered those templates a long time ago. So step one is to get updated versions of all the templates you own, wherever possible.

Open an unconverted V6 APP via the Open Solution button on the Start Page as per above. You may see a dialog window telling you that some of your WINDOW or REPORT structures have some errors (refer figure [4 errors found conversion windows]). I'll come back to those in Part 2. In the meantime, press OK to dismiss the dialog. AppGen will read your V6 APP. If there are any problems during the process it will write corresponding messages to the Error window and when finished it will display a dialog giving you the choice of whether or not to save the converted APP. Click on No. If you click on Yes, AppGen will proceed to load the broken-but-converted APP but the IDE will clear the Error window which is definitely not what you want.

If the Error window is retracted, press Ctrl+Alt+K (View|Errors) or hover over the Errors tab to extend it. (If you use the keyboard, make sure the mouse is not hovering over one of the adjacent tabs at the time. You may need to click in the main pane for the keypress to work.) Click in the Error window, press Ctrl+A to select all messages, press Ctrl+C to copy the selection to the clipboard, paste the clipboard into a window in your favourite text editor and finally save the text file in your text editor. AppGen writes messages to the top line in the Error window so the text file you've saved should be read from the bottom up.

You may have multiple "Unknown template type" messages but when you look at the template family in the parentheses at the end of each line you'll find only a small number of individual template sets mentioned multiple times each, once for each occurrence of each problematic template as they are encountered. As a result, you may only need to register a small number of template sets to prevent all the messages on the next run.

There will also be a "Cannot save converted application" message at the top of the file, being the most recent message, which was emitted when you answered No to saving the broken-but-converted APP. This message can be ignored.

Close any open APPs and choose Tools|Edit Template Registry menu command as per above. Starting at the bottom of the message file, read the name of the template set in the parentheses and register that set as per above. If a particular template set is missing from the SoftVelocity or accessory template directory, you will need to copy/install it before attempting to Register it. If you've already added the template set, double check the name of the template appears in the template tree.

If you're not sure of the name of the template file(s) you need for a particular template set, perform a text search within multiple files for the text in the parentheses in the message file – this text is the name of the templates set and will appear in a TPL file in a line beginning with the text #TEMPLATE(<name_of_template_set>, ... Look in this TPL file for any lines beginning with #INCLUDE('<name_of_file>') – they will tell you which other files you need. You should also look for any lines beginning with #HELP('<name_of_file>') – help files are not mandatory but if they're missing then clicking on a Help button within that template set will not achieve anything.

Repeat the registration process for each unique set name.

Click on the Save and Exit icon on the Template Registry toolbar.

Click on Open Solution button on the Start Page. Browse to and open the unconverted V6 APP that yielded the broken conversion last time. Now that all the required templates are registered, the attempted conversion should go further

this time.

If you still receive some "Unknown template type" errors and you're sure the correct files have been registered with the Template Registry, particularly in the case of templates you've written yourself, double check that the name of a problematic template set hasn't been changed since the last time it was registered.

At this point, you should have an APP loaded in the IDE, which will also have created an SLN file and a CWPROJ file in the same directory as your APP file.

Export the application to text (Application|Export Application to Text) and name the output file <name_of_application>.TXA_BROKEN. Using this naming convention lays the foundation for before and after versions of the TXA file. When you make all the fixes you save a second TXA as <name_of_application>.TXA_FIXED – the differences between the two TXA files (which you can get via your favorite differences program) gives you a checklist of window/report controls to be verified during the port. That is, you will want to look at each control at design time and at runtime and verify the control is behaving the same way as it did before the port

Open the resulting TXA_BROKEN in a text editor or in the IDE (File|Open File...). Search for !!> ERROR(. If your editor supports it, perform a search that lists all occurrences in an output window – this will give you a good overview of the errors to be fixed.

In the IDE, click on the Search Results tab (or choose View|Tools|Search Results). Double click on the first error to take you to the line in the TXA_BROKEN file. Scroll up to see the name of the structure containing the error (DETAIL or REPORT or WINDOW structure in a [WINDOW] section and search backwards for [PROCEDURE] to see the name of the procedure (listed after NAME in the [PROCEDURE] section.

In Part 2 I'll go on to window and report errors.

Philip Prohm is Director of Optimate Group Pty Ltd, a holding company for numerous ventures including Optimate Computer Systems, Broad Ribbon Designs and The International Camelid Register. The group has customers in five countries. Optimate Computer Systems was established in 1991 in the days of Clarion 2.0 for DOS and these days is Australia's leading vendor of alpaca breeding software, a shrinkwrap application written in Clarion. Away from computers, Philip plays (field) hockey and is partial to science fiction. He and his lovely wife and their two beautiful children are based in Wollongong NSW where they are busily building their global empire.

Reader Comments

Add a comment

Clarion Magazine

Converting C6 Apps to C7, Part 2

by Philip Prohm

Published 2010-03-11

In Part 1 I dealt with dictionary and template problems. Now it's time to go on to window and report errors.

WINDOW/REPORT Errors

Click on the Application Tree tab if it isn't already active and scroll down the list of procedures looking for a tiny warning symbol, same as the icon on the Warnings button on the Errors window only smaller.

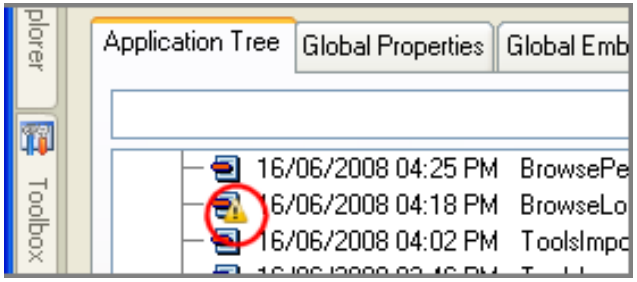


Figure 1. Warning indicator

When you've found a problematic procedure, save the APP (press Ctrl+S or click on the Accept changes AppGen toolbar blue floppy disc icon) because the next step may cause the IDE to enter an infinite loop.

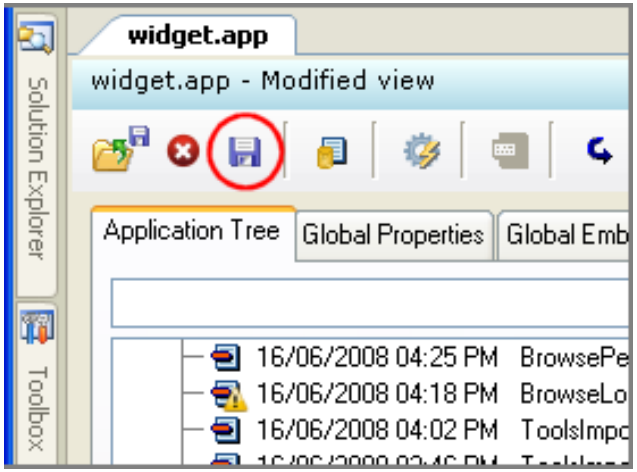


Figure 2. Saving the changes

As above, export the procedure to text (Application|Selective Export), but this time name the output file <name_of_procedure>.TXA_BROKEN. When you fix the errors in a procedure in the IDE the warning icon is removed so you lose the visual flag for which procedures you should be paying closer attention to in the process of porting your APP to V7. When you make some fixes you save a second TXA as <name_of_procedure>.TXA_FIXED, and you can again use your favourite differences program to view the changes.

Also, make sure the columns in the Errors window are wide enough to be useful, particularly the Description column. If you have errors that put the IDE into a loop, you won't be able to widen the columns at that point. Also make sure the Errors window is tall enough. Of course, first up you won't know how tall is tall enough but the second time through you know the IDE will hang for the particular procedure so you can go ahead and make the Errors window very tall leaving only a few of lines of procedures visible in the Application Tree tab, and the Window button visible in the AppGen window. Highlight the problematic procedure then click on Window button.

If the IDE hangs, take a screendump of the Errors window or type the error text into a text file for later reference, restart the IDE and reopen the solution (eg. click on the name of the solution from the Start Page).

Recent versions of the IDE may prompt you whether you want to use the last saved version of the APP or whether it should attempt to recover unsaved edits. Recover works well but so does Last Saved if it is sufficiently recent (and can be faster) so use your own judgement here.

Export the procedure to text (Application|Selective Export) and open the resulting TXA in a text editor. Scroll to the bottom

to the [WINDOW] section and there you will see the same text you would see if you'd been able to click on the Window button directly. Look for !!> ERROR(n) and take action in the APP to remove the source of the error. One possible action is to edit the TXA and import the modified TXA into the APP. For example, if the error is "The #SEQ attribute is incorrect" (see below) you could delete the ,#SEQ(n) text from the TXA file.

An example of an error that can cause AppGen to enter an infinite loop is <name_of_procedure> Error: No Window Defined! although it is also possible and more common to have this error without having AppGen caught in a loop. A solution to the problem where AppGen is indeed stuck in a loop is to copy the TXA file and edit the copy (giving you a before and after for later reference). Fix the window errors (!!> ERROR(n)) and import the fixed TXA into the APP (Application|Import Txa). You may find that the warning icon is removed completely from the procedure name, leaving you no indication that the procedure is suspect, so the existence of the pair of TXAs you made serve as a checklist for procedures to be checked such that their behaviour is unchanged in V7 compared to V6. Moreover, the differences between the two TXAs serve as a checklist for window controls to which to pay particular attention, as I mentioned earlier.

Another example is ASSERT: %FixClassName: Cannot find class <name_of_class>. A solution to this problem is to ensure the relevant source files are in the libsrc directory (commonly C:\Program Files\SoftVelocity\Clarion71\LibSrc\win). When copying template files to the template directory (commonly C:\Program Files\SoftVelocity\Clarion71\template) it is easy to overlook the accompanying source files.

From the error information in the TXA file plus the error information from the Errors window you can decide which errors to tackle first for that procedure.

If the WINDOW structure has no syntax problems, you will be taken to the Window Designer, in which case you should cancel your way out of the Window Designer and click on the Report button and skip the following section. If, however, the WINDOW structure does have a syntax problem you will be taken to the Properties page instead of the Window Designer (or in the case of early releases of V7, the Window Structure Source Editor) and the Window Designer button will be disabled.

Click on the ellipsis button to invoke the Window Structure Source Editor.

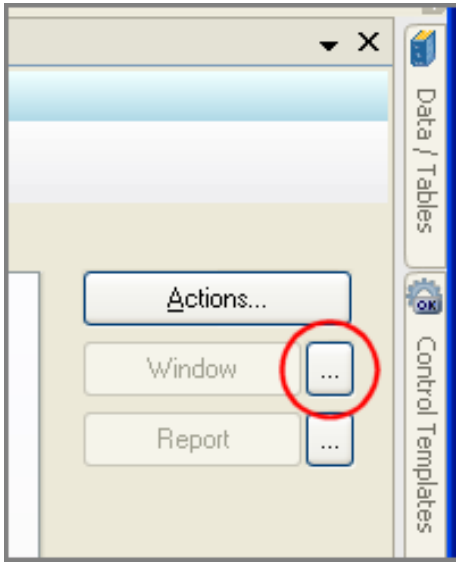


Figure 3. Invoking the structure editor

Look for a line beginning with !!> ERROR (n)– *n* is the column number of the error. There are a variety of possible errors you may see at this point. Here are some examples along with some suggestions for fixes:

!!> ERROR(): The #SEQ attribute is incorrect or refers to wrong control template instance

Remove ,SEQ(n). Make a note of the control because you might need to add the associated control template to the window. Later when the window/report is okay, check if a control template needs to be added. If the control template adds an equivalent control you will probably want to remove the old control and you might need to fix up any references to the old control so that they refer to the new control. You may also need to recreate any embeds associated with the control - again, comparing your TXAs and/or generated source will tell you if this needs to be done.

!!> ERROR(): Value of parameter has incorrect type

You might find the third parameter of the USE attribute is lacking a question mark ("?"). This parameter is supposed to be a field equate label so prefix the parameter with a question mark and that will fix it.

Don't forget to remove the !!> ERROR (n) line. If the editor won't let you Save and Exit then it has detected something it still doesn't like about the structure so keep looking for a syntax problem. If you're convinced there is no syntax problem, abandon your changes and close and reopen the entire application and make the changes to the procedure again –

this time the procedure will save without incident.

For report procedures, the next step is to repeat the above process for the REPORT structure – click on the ellipsis button next to the disabled Report button to invoke the Report Structure Editor.

Fix the errors and you're done here. The online help also has some information; see Help|Context Help|Contents|Getting Started|Resolving Conversion Issues from C6 to C7 Applications.

Since you're porting an old app to Clarion 7 you probably don't have any source files, in which case you'll want to generate all sources from scratch. The Generate the currently selected Application icon on the menu toolbar won't do that if the settings on the Applications Pad aren't appropriate. The most foolproof way is to simply press Ctrl+M (Build|Generate and Make Current Application).

Figure 4. Generate the application

If you want to use the toolbar then open the Applications Pad (hover over the Applications tab on the left window edge or choose View|Applications from the menu) and click on the down arrow adjacent to and to the right of the lightning bolt (second from the left below the "Applications" pad window title) and make sure the settings are set to Generate Selected and Unconditional Generation.

Notice that the text in the shaded Description pane changes to Generate Selected Apps Unconditional Generation – this text will always be visible when the Applications Pad is open provided the pane is large enough (it's physically possible to resize it too small to display text) so you will be able to see the settings at a glance without having to click on the down arrow.

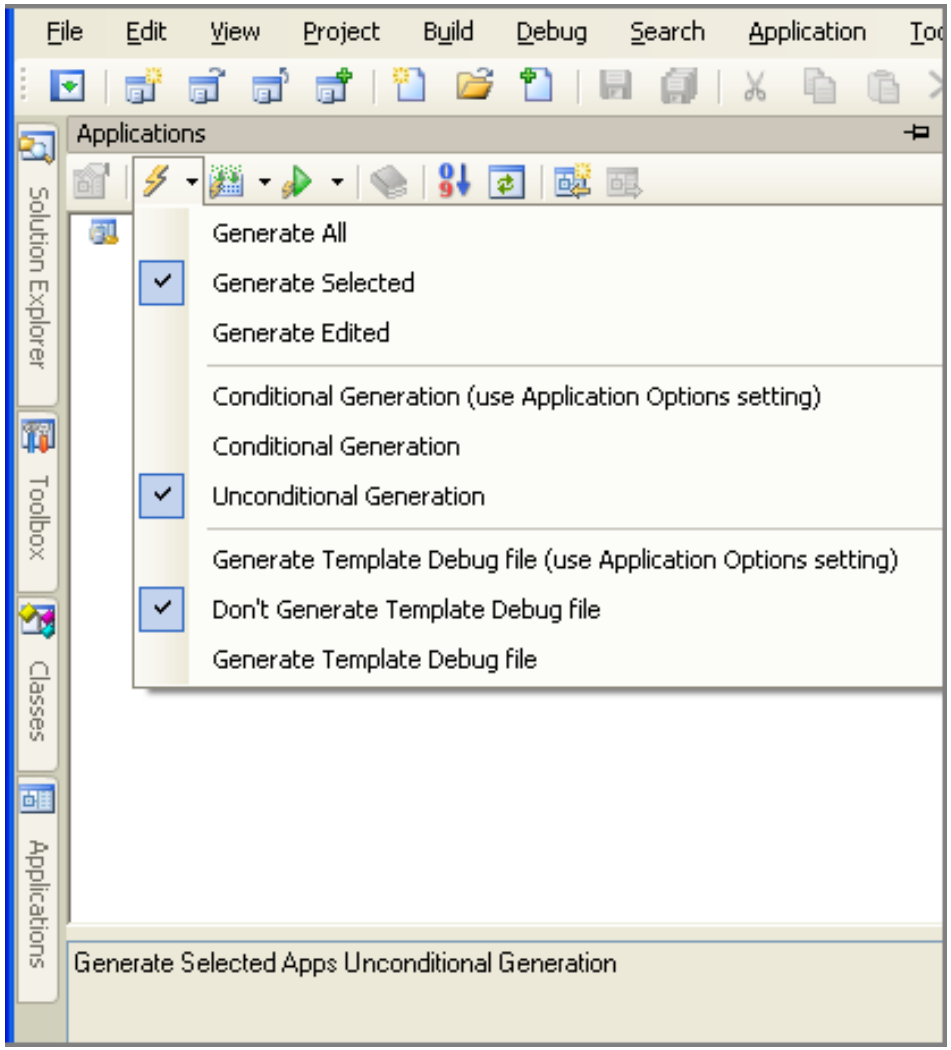


Figure 5. Applications Pad code generation settings

Together, these two settings will make the Generate and Make Current Application toolbar icon act like Ctrl+M; you can relax the settings later when you wish (refer to Figure [5 generate and make]). The setting for the Template Debug file is unimportant for purposes of this discussion.

When Generate and Make is finished the IDE will parse your APP file. If the APP is large, this will take some time so you might like to disable parsing until later when you start doing some new development. Parsing gives you things like code completion which you don't need at this stage of porting your app. Go to Tools|Options|Clarion|Enable project parsing and turn it off.

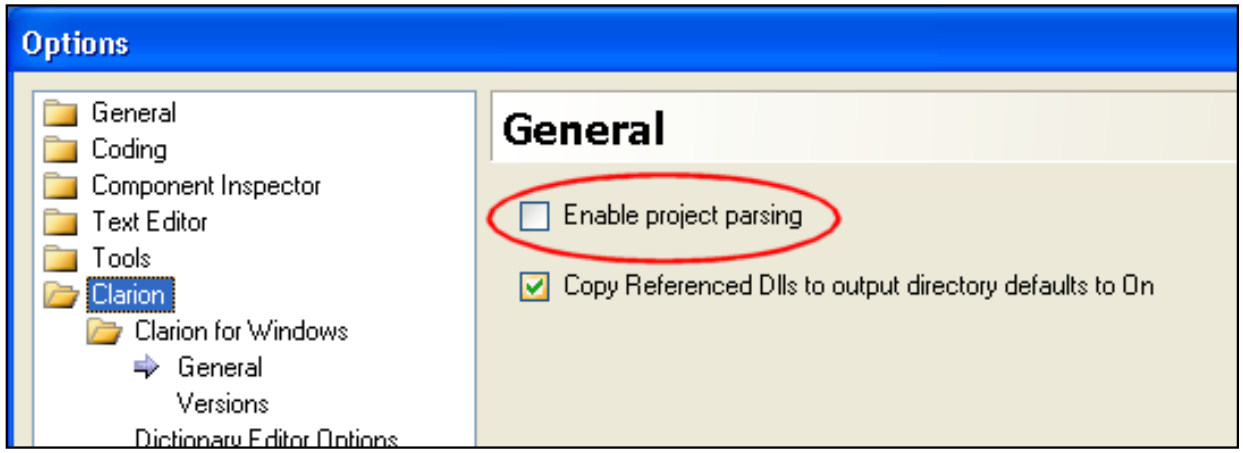


Figure 6. Disable project parsing

At this point you may have some compilation errors. Some of those errors may be due to template sets for which you haven't actually copied all the source files to the appropriate location, in which case do that now.

You might find that some of your own handwritten source files can't be found by the compiler. Perhaps you didn't copy all files to the right places or perhaps the V7 redirection file isn't as fully set up as you thought. To fix the redirection file you can open it via Tools|Edit base redirection file|(Current Version).

Perhaps you have written a library of source files that are used by more than one of your programs, and one of your library files is now apparently incompatible with the V7 compiler. While the way you deal with individual issues of incompatibility is clearly up to you, one possibility is to use conditional compilation so that both the old compiler and the new compiler are happy.

For example, let's say in one of your library files you refer to EVENT:VBXevent which is defined in EQUATES.CLW in Clarion 6 but not in Clarion 7. Sandwiching the reference in an OMIT compiler directive will keep the V7 compiler happy without breaking any code compiled with the V6 compiler:

```
omit('endo/', _C70_)
  of EVENT:VBXevent        ;    str='EVENT:VBXevent'
  endo/
```

Clarion 6 doesn't know what _C70_ is so it doesn't omit that code. However, in Clarion 7 and later, _C70_ is a flag defined as ON so those compilers do omit that code. Of course, you will need to check the source code you are porting to V7 to ensure it doesn't refer to that event or that you can OMIT those references.

As I mentioned earlier, if any of your dictionary columns were renamed when you imported the dictionary, references in your embed point code to the old name of the column will be invalid. Click on the error message in the Errors window to take you to the relevant embed point where you can then change the name of the column to the new name.

Clarion 7 splits lines slightly shorter than Clarion 6 so if you use any templates that generate long lines in comments you might find that the last part of the line is now on a new line of its own unprotected by a comment character and hence is now interpreted by the compiler as code (or data, depending on the physical location in the source file) yielding a compile error.

If you edit a template without editing a module you will have to set code generation to be Unconditional otherwise the module will not be generated according to the newer version of the template. Of course, you will have to reregister the changed template and the easiest way to do that is to close and reopen the APP. The toolbar Generate button won't do unconditional generation so you have to use the Applications Pad Generate button.

At this point you should have fixed all the missing symbols.

Perhaps you have duplicate symbols now.

If any duplicates are related to a third party template in your APP, check with the vendor for an update for Clarion 7 – the fix might be as simple as installing the update and recompiling the APP.

Perhaps SoftVelocity has added a new symbol that clashes with an existing symbol in either your code or a third party vendor's code. Or even perhaps SoftVelocity and a third party vendor have both added an identical symbol to their respective Clarion 7.x releases. Search source files and template files for occurrences of the text string in the error message. The messages are quite informative, usually being of the form:

> Duplicate symbol: <symbol> in <objectfile1>.obj, <objectfile2>.obj

so straight away you get a pretty good indication of the nature of the clash. Sometimes they are only of the form:

> Duplicate symbol: <symbol>

which clearly is less helpful. These tend to refer to Windows API functions. I'll return to this scenario in a moment.

Looking at the first form, there are a couple of starting points. Firstly, the object files are named after the source files, so from this you know which source files to search. For example, if <objectfile1>.obj was ABFILE.obj then you search ABFILE.CLW.

Secondly, the basenames of these files often indicate their origin. For example, filenames of the form AB<something> are usually SoftVelocity ABC files. Other vendors often follow a similar convention using a two- or three-letter "signature" at the beginnings of their filenames. Using this information you can often nail down the nature of the symbol conflict pretty quickly.

If you can't obtain an update from a vendor, here is something you can do.

First, choose the non-SoftVelocity prefix as the file set to edit. You can edit the SoftVelocity files but if you later update your Clarion version you'll at worst clobber your edits or at best have to merge them with the files in the new version. Since you're expending time and perhaps other resources porting an existing program to Clarion 7 there's a reasonable chance you will one day update your Clarion version.

Second, in the file set, identify the individual files to be edited. It may only be one file but you need to locate each file before you edit it. One way is to employ a text editor that can search for text in multiple files. These editors usually have a results window that lists for each matched string the full path of the file and the line number on which the string was found.

In the IDE, do it like this:

1. Press Ctrl+F (or choose Search|Find).
2. In "Find what:" enter the search string (see below).
3. In "Look in:" type in or browse to <clarionroot> (eg. C:\Program iles\SoftVelocity\Clarion71).
4. Tick (check) the Include sub-folders option. You need to do it this way because you need to search both <clarionroot> \libsrc and <clarionroot>\accessory\libsrc.
5. In "Look at these file types:" type in "*.clw; *.inc". For an exhaustive search you can use "*.clw; *.inc; *.equ; *.trn; *.int".
6. Clear "Match case" and "Match whole word".
7. Set "Use" to "Standard search".
8. Click "Find all".

In the Search Results window you will have the matches found. Doubleclicking on an entry in the Search Results window will take you to the line of the file of the match. At this point you probably won't want to do this because you're only identifying which actual files need modifying but it's a handy thing to know and since we're here now is the time to at least mention it.

The string to search for is usually the <symbol> in the error message. However, the duplicated symbol may be prefixed by an attribute so you can't always use it literally. For example, in ABFILE.CLW is a QUEUE called RelationQueue which is a type definition.

```
RelationQueue        QUEUE, TYPE
                       ...
                     END
```

If someone wrote a second file which also had a RelationQueue QUEUE,TYPE in it, the <symbol> in the duplicate symbol message would be TYPE$RELATIONQUEUE in which case you ignore the TYPE$ and search the file set for RelationQueue. Clarion is a case-insensitive language so your search should be case-insensitive. Editors usually default to case-insensitive searches in which case there's nothing special for you to do.

Perform the search, identify the specific file, locate the physical file in the file system of your hard disc (e.g. using Windows Explorer), copy the file and rename it such that the new name is the old name plus an identifying suffix. For example, let's say the vendor is Fabulous Software; the initials are F. S. so you invent an identifying suffix of FS. Or perhaps they invented that for themselves already and name their files FS<something>.CLW. Either way, let's pretend the name of the file containing the duplicated symbol is FSFILE.CLW – rename the copy to FSFILE.CLW.FS.

This convention allows to you to modify FSFILE.CLW as much as you like while retaining a permanent copy of the vendor's original.

When you've finished a batch of modifications to FSFILE.CLW, copy it to FSFILE.CLW.<your_initials>. Your initials can be your personal initials or the initials of your company, whether you own it or are employed by it. If the initials you choose are PP then copy FSFILE.CLW to FSFILE.CLW.PP. Now you have three files, FSFILE.CLW, FSFILE.CLW.FS and FSFILE.CLW.PP.

This convention allows you to chop and change versions of FSFILE.CLW at your convenience. If you want the compiler to

use the original version, copy FSFILE.CLW.FS to FSFILE.CLW. If you then want the compiler to use your modified version, copy FSFILE.CLW.PP to FSFILE.CLW. You no longer care about FSFILE.CLW being clobbered, and you can always tell immediately from the file system which FSFILE.CLW is being used simply by comparing the size and/or timestamp of FSFILE.CLW with the size and/or timestamp of FSFILE.CLW.FS and FSFILE.CLW.PP.

Further, any time you need to understand the changes you've made to the file, run differences on FSFILE.CLW.FS and FSFILE.CLW.PP and the output will tell you exactly what the differences are.

I recommend you bookend your modifications with comments. A differences program will usually show up the changes pretty well but when you're actually editing a file it's very handy to have clear delimiters before and after your modification, and to have a clear record of the original code. For example:

Before:

```
RelationQueue       QUEUE, TYPE
                      ...
                    END
```

After:

```
!pp 15-1-10   clashes with SV ABC RelationQueue
!RelationQueue       QUEUE, TYPE
RelationQueueType    QUEUE, TYPE
!pp
                      ...
                    END
```

In this example, !pp <date> <desc> and !pp are the bookends. In template language files, where the comment symbol is #!, you would use #! pp <date> <desc> and #! pp. Date and especially description are optional. Use your judgement but with so many text editors allowing you to paste the current date at the cursor position with a single keystroke, it's difficult to mount a case for omitting the date – the date is just too useful for reference when you come back to examine the code in the future.

Lib files

Compiled libraries (LIB and DLL files) are another potential source of conflict. Perhaps you have a duplicate Windows API symbol such as GetFileVersionInfoA or VerInstallFileA. In Clarion 7, SoftVelocity greatly increased the number of Windows API calls included in its libraries, so the chance for a conflict is much higher. If you get a conflict you may not need the old library at all. You can identify the library by searching files for occurrences of the symbol.

To delete a problematic library, use the Solution Explorer (Solution Explorer|Solution widget|widget|Libraries, Objects and Resources|version.lib|delete-key).

At some point you will actually compile!

Conclusion

Clarion 7 is a huge change from previous versions of Clarion, and the migration path isn't always easy, but with a little perseverance and the techniques I've outlined you should find it manageable.

The key to a successful migration is to compare, compare, compare. Above all compare your C6 and C7 generated source to make sure all your code made it across as intended.

There is, of course, one other important step, which is to ensure that the runtime behavior is the same in C7. That's a subject for another article ....

Philip Prohm is Director of Optimate Group Pty Ltd, a holding company for numerous ventures including Optimate Computer Systems, Broad Ribbon Designs and The International Camelid Register. The group has customers in five countries. Optimate Computer Systems was established in 1991 in the days of Clarion 2.0 for DOS and these days is Australia's leading vendor of alpaca breeding software, a shrinkwrap application written in Clarion. Away from computers, Philip plays (field) hockey and is partial to science fiction. He and his lovely wife and their two beautiful children are based in Wollongong NSW where they are busily building their global empire.

Reader Comments

Posted on Thursday, March 11, 2010 by Robert Wagner

I was lost after the first 2 sentences.

Export the app? The IDE might go into a loop?

Those are bugs in the IDE. In fact, most of what the article mentioned are C7 bugs. Workarounds are fine if an app only has 10 or 20 procedures, and only a few files. But if you have thousands of procedures, and hundreds of files, I can't imagine that conversion to C7 is a prudent course of action. The C7 conversion program should take care of 100% of the changes that are required to migrate to C7; anything less is not acceptable.

Seems to me that SV did two things at the same time, when only one should have been done. The IDE was a major change. But then they also changed the run time. So SV was aiming at a moving target, with a weapon that explodes.

The easy thing (and maybe it's still the easiest thing) would have been to rewrite those portions of the C6 IDE that prevented 32 bit operation. That's not hindsight, either. Over 10 years ago, something along those lines was touted by Barrington. And once you had the IDE jerked into the 20th century, you could, as a separate thing, upgrade the run time. And you could have collected money for both upgrades - they could have been C6 (32 bit IDE), and C7 (upgraded runtime).

.....

Posted on Sunday, March 14, 2010 by Russell Eggen

Robert,

< text snipped >

A concept to keep in mind is the state of C7 when Phil penned the article. Softvelocity is publishing more or less regular builds and the issues he found may no longer exist.

I think the most sensible approach is to correct things on the C6 side and then try the conversion again. Repeat until all of your apps convert 100% error free.

One aspect Phil covered that I did not, is code that is C6 only and his OMIT solution is excellent.

.....

Posted on Monday, March 15, 2010 by Philip Prohm

Hi Robert,

The article (split into two parts) is a practical guide and contains some solutions to some problems that might arise. I hope they are helpful to people but in your case perhaps they aren't, for which I'm sorry.

Philip

Posted on Monday, March 15, 2010 by Robert Wagner

Hi Philip:

Your articles were great! They helped me, and I am sure they helped many others in the community. Without folks such as you the Clarion community would be greatly diminished.

My rant was targeted at the state of C7. I'd give my eye teeth for a good, solid, C7 that I could convert to. The limitations of C6 (basically, the limits created by 16 bit modules) have made development difficult in C6 for some of my apps, and that is frustrating.

Keep up the good work. Clarion Magazine and writers such as yourself are the great strengths of the Clarion community.

Add a comment

Clarion Magazine

Where Delete Occurs, Part 1

by Steven Parker

Published 2010-03-23

Life was so much easier before ABC and Edit-in-Place and allowing users to delete records directly from the browse.

In the past, all CRUD operations took place on a form. From the browse, the user pressed a button or key, the form appeared and the user completed (or canceled) the action. The Form template handled I/O. It was predictable. It was programmable. By me.

Whether I am creating ABC or legacy apps, whether or not I use Edit-in-Place, the default behavior for a Clarion Win32 app, especially on delete, has changed. The default (and, now, expected) behavior is not to show the form.

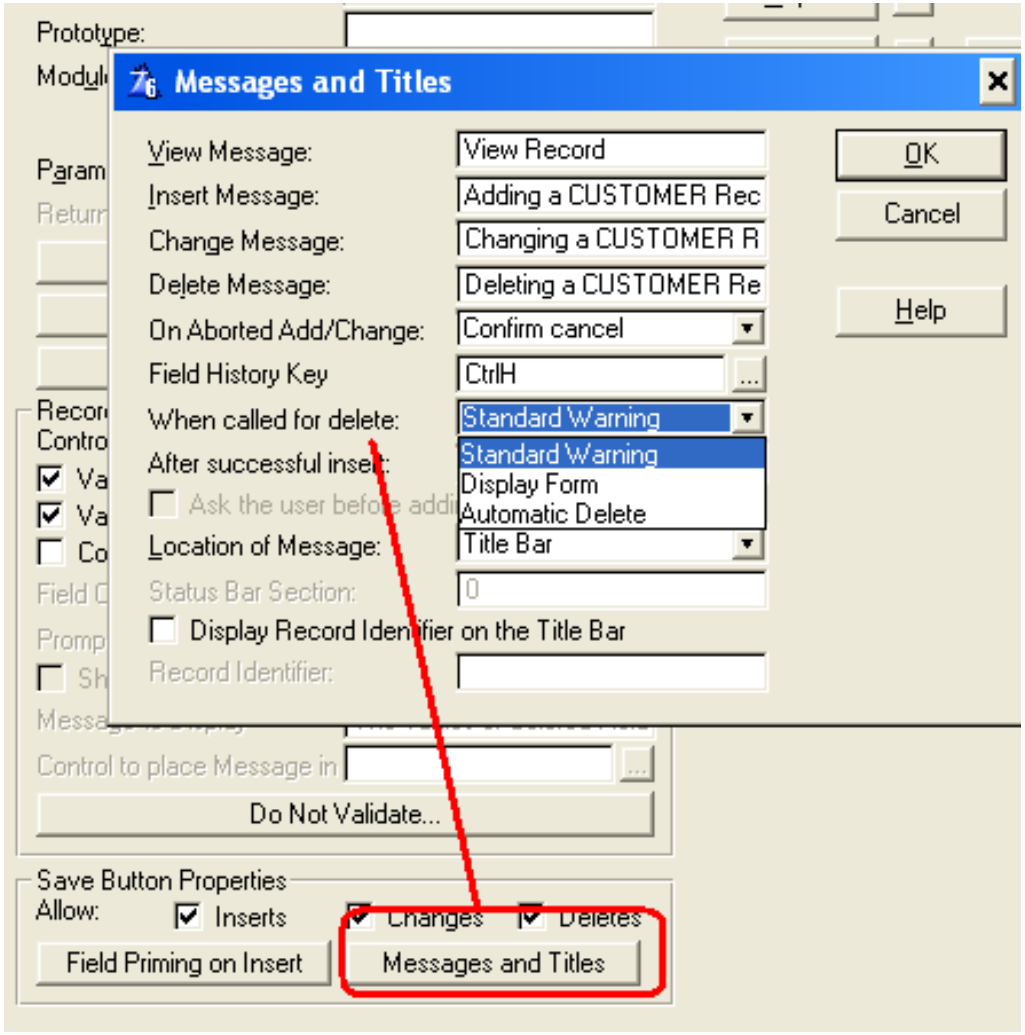


Figure 1. Form Properties worksheet for "action when deleting"

If I do not change the form's Messages and Titles, my users will get the "Are you sure you want to delete ...?" message. But, while this is the default behavior, I *can* change it (as shown in Figure 1, above). Automatic Delete, which I do not recall ever using, doesn't present the confirmation message, the record is just deleted.

This has implications for the developer. It could mean that the browse controls part of the action (as it must, to support EIP). Should I need to customize the handling of the delete process, I may not be able to embed code in the form. Or, worse, not in the form *only*.

It is not often that I need to customize delete ops. But there are times that I want to and, when I do, the standard behaviors available through the template must be circumvented.

Common Customizations

Note: I will talk a lot about deleting but wherever deletes occur, inserts and updates occur. So customized handling of *any* I/O op can be done if I identify where any *one* of them happens.

Delete or Flag: Suppose I want to mark records "deleted" or "inactive" instead of actually deleting them? Suppose I want to offer the user the choice to flag the record or delete it?

Post-processing: Suppose I need to do something special after a delete is confirmed, after it actually happens. Perhaps my customer wants a log entry that something was deleted. Or, if the user deletes something, the user wants to automatically run a report. Who knows? Customers are so creative in their desires.

Managing RI: How and where delete occurs becomes very important when related files exist. That is, delete operations create additional concerns for Referential Integrity.

Why is RI an issue? After all, I can specify RI constraints right in the dictionary. I can let the templates handle RI for me:

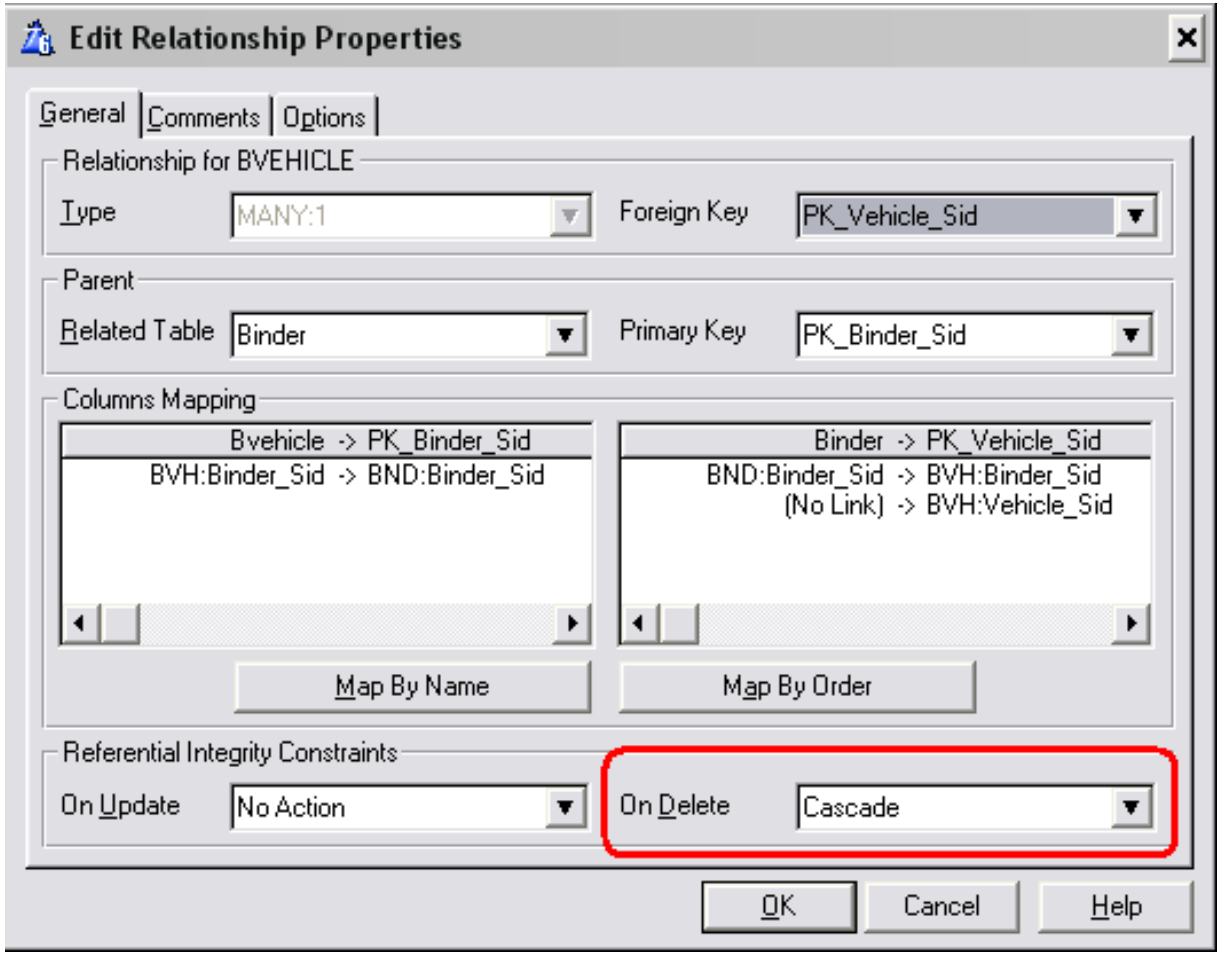


Figure 2. Set RI constraints in the dictionary

Cascade Deletes

By telling the dictionary I want to cascade deletes, the templates – I don't know whether it's the Browse template or the Form template and I don't care – generate the RI code for me. The generated code has to accomplish something like:

```
Child:KeyValue = Parent:KeyValue
Set(Child:ParentKey,Child:ParentKey)
Loop while Child:KeyValue = Parent:KeyValue
```

```
    If ErrorCode()
      Break
    End
    Access:ChildFile.Delete()
  End
```

What could be easier than not writing any code at all? (However, if I *did* care about the template generated code, I'd probably look at RelationManger.DeleteSecondary in abfile.clw.)

But, this kind of code can be very slow. Suppose I am deleting an inventory item, not only can Inventory have multiple child files but those files can have thousands of records.

Existence of the Inventory item in some child files should be Restricted, not allowing the inventory item to be deleted. Purchase Orders and Bills of Materials are excellent examples of this situation. But Customer (item) Sales and Sales History can each have thousands of records for each inventory item.

I've seen cascading deletes of an Inventory item take 10 minutes. That was running locally, no network. Deleting a customer and her/his A/R and sales history records was slower still. Imagine hundreds or thousands (my customer accidentally set 14,000 customer records for delete!)....

In case of SQL

Run across a network, with many records, cascading deletes can become painfully slow.

Further, if my back end is SQL, looping through records is just plain dumb.

In SQL, I could create a stored procedure or I could create a database trigger. But

- creating either a stored procedure or a trigger in the database requires a skill I do not happen to possess
- the DBA may very well not allow me to modify her/his database and, besides, even if s/he does allow it, I have no guarantee that, on a database update, my code will still be there (DBA's *are* agents of the devil, after all)

Even if I control the database, I still need to know where delete occurs in order to call my stored procedure at the right time.

Dictionary triggers

If all I want to do is handle RI deletes myself, against a SQL back end, the Dictionary Editor supplies the tool I need. This tools is the ability to declare triggers in the dictionary. These are not triggers in the "normal"sense of the word, since they don't automatically execute on the *server* when a particular action (such as a delete) is performed – call them "Clarion client-side triggers." But they accomplish the same result.

From the Dictionary Editor, left click on the file/table name. The Triggers option is third on the list:

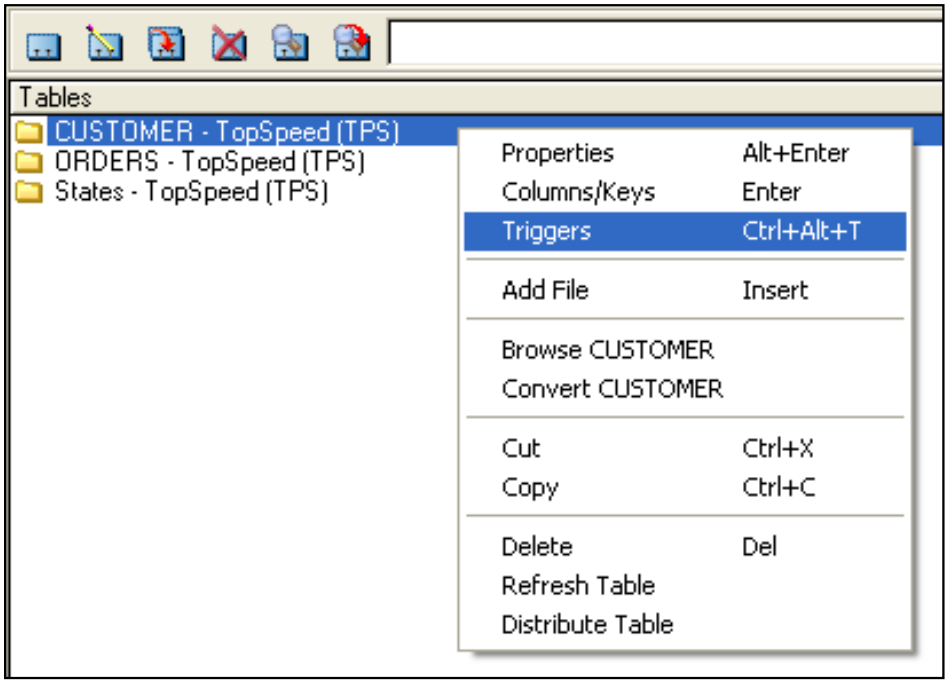


Figure 3. Dictionary triggers

I can select before or after any I/O operation:

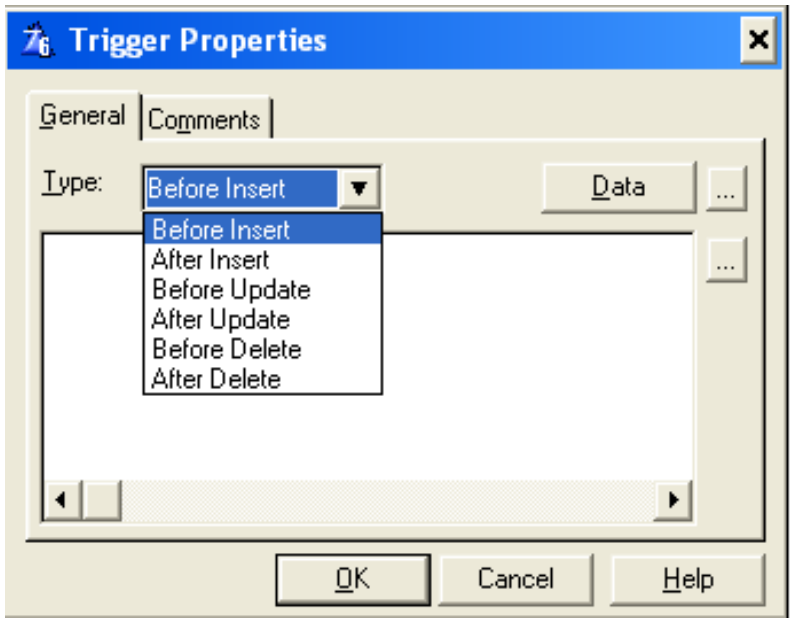


Figure 4. Available triggers

I only have to enter the desired code:

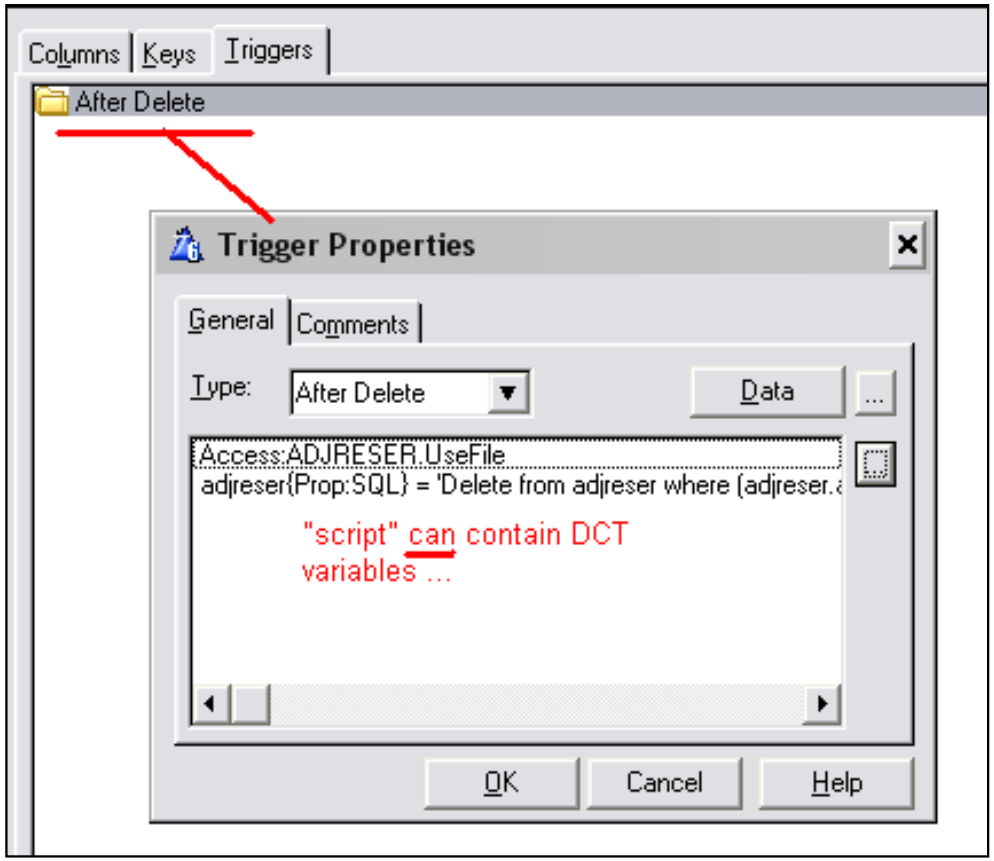


Figure 5. Dictionary trigger declaration

The full code in this trigger is:

```
Access:ADJRESER.UseFile
adjreser{Prop:SQL} = 'Delete from adjreser where ' & |
  (adjreser.adjusterno = ' & ADJ:AdjusterNo & ');'
```

Note that I can use dictionary variables in the trigger. (I'm not certain that the Access:ADJRESER.UseFile is necessary but I don't expect it hurts either.)

I could, I suspect, also have used:

```
Relate:ADJUSTER.Delete(0)
```

for my trigger. I don't know if this would offer any advantage with TPS files (it certainly offers none for SQL).

The advantage of this for SQL back ends, however, is obvious and is ample justification for not using template generated RI code in all circumstances.

Non-SQL databases

I think I have made a good case for not using dictionary/template cascade deletes for SQL databases. And, for SQL databases, I have provided a more than acceptable dictionary alternative. And, better, I don't have figure out where delete occurs.

But this does not mean that I am stuck with standard cascade deletes for TPS databases.

The Prop:SQL I used in the dictionary trigger got me thinking about having work done elsewhere. Issuing a SQL query (which is what Prop:SQL does), in essence, starts some work to be done in the database, not inside my program; that is, somewhere else.

Clarion has a statement that allows me to do work elsewhere: Start. I can Start a procedure, passing it sufficient information to uniquely identify the record I want to delete:

```
Start(DeleteCustomer,,CUS:CustomerNumber)
```

The DeleteCustomer procedure would look like this:

```
Clear(CUS:Record)
Open(CUSTOMER,42h)
CUS:CustomerNumber = pCustomerNumber
Get(CUSTOMER,CUS:CustNumKey)
If ~ErrorCode()
  Relate:CUSTOMER.Delete(0)
End
```

```
Close(CUSTOMER)
```

This procedure fetches the CUSTOMER record and, if the customer record is found, does the cascade delete.

Note: If you use a STARTed thread without a window, you need to make sure the thread exits or else your program will stay in memory after being closed by the user There will be no visual indication that there is an orphaned thread still running.

Unfortunately, to use this technique means that

- I *do* need to know where the standard delete action occurs
- I need to bypass the standard delete (or the standard Relate:CUSTOMER.Delete will occur and that is precisely what I want to avoid)

By STARTing my delete procedure on a new thread, the browse from which I called the delete stays responsive, not waiting on the deletes to finish. Since I have used this technique, I can say that control returns to the browse much sooner than waiting for the standard deletes.

> **Note:** START isn't the solution to every responsiveness problem. Remember that you're exchanging control over *exactly when* the delete happens in exchange for a better user experience.

If I turn off cascade delete in the dictionary, I can't use Relate:CUSTOMER.Delete. But, I can START one or more procedures in which I manual do my deletes, one child file at a time. This, I can tell you from experience, also works.

One solution is not enough

If RI constraints, specifically cascading deletes to child files, were my only concern, I'd be done. Dictionary triggers are an ideal way to handle cascading deletes.

Well, no, I'd be done if I only used SQL.

And only if I was concerned with deletes and no other file operations. And, no other user options....

But because I often use TPS databases, I may find myself in a position where I want to do my own RI. I also still have the need to be able to flag a record instead of deleting it and the need to "ask flag or delete." Each of these requires that I

know where delete occurs and how to bypass, under program control, standard delete action.

So, I still need to know where delete occurs. Oh, yes, it should be the same place that inserts and updates occur.

I'll cover that next time.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Where Delete Occurs, Part 2

by Steven Parker

Published 2010-03-23

In Where Delete Occurs, Part 1, I persuaded myself that I needed to know where, exactly, deletes (and, therefore, inserts and updates) occur. I need to know so that I can subject these operations to program control.

Among other things, I may want to be able to:

1. conditionally enable/disable CRUD operations
2. flag an item instead of deleting it (optionally filtering the browse on the flag)
3. ask the user whether to flag an item or delete it
4. perform additional operations after a user confirms a delete op

and perhaps most importantly,

5. handle cascading RI updates or deletes myself, manually

Where Delete Occurs, Part 1, I did provide what I think are excellent solutions for cascading deletes for SQL and for flat file systems. These solutions are also adaptable, I think, to cascading updates (though using a dictionary trigger to cascade a change in a single field – dictionary triggers are not available at the column/field level – might be a bit more work; of course, if you find yourself needing to update a link field, you may want to reconsider your database design).

So, then, where *does* delete happen?

The process of discovery

There a a number of ways I could discover what I want to know.

> **Note:** The demo apps that accompany this article (downloadable at the end of this article) are based on the Tutor.APP from the Examples sub-directory of Clarion 6 (dictionary triggers are not available in earlier versions of Clarion). I am including the original Tutor.APP and data files, each in separate archives. The demo for this section is also called Tutor.APP. The EXE is provided locally linked.

The most obvious strategy is to compile a small application in debug mode and trace a CRUD op step by step. I would be looking for any of the ABC I/O methods. However, plowing through the generated objects seeking an Insert, Update or Delete gives the impression of being a bit ... time consuming.

Alternately, I could put Messages throughout the embeds of the browse and form. As each message appears, I can check the file (let's assume I'm using TPS files so that I can check the file fairly easily using TopScan) or display Records(<*file*>) in the Message.

This, too, seems like might take a while.

Or, I could simply check my knowledge base. Searching for the keyword "delete," I would discover that others have already done the research for me and that delete happens in ThisWindow.PrimeUpdate.

The PrimeUpdate method is found in abwindow.clw, around line 853. It is a WindowManager method.

Examining the code in PrimeUpdate clearly shows that this method handles inserts (field priming) and deletes (the actual delete appears to be done c. line 881). It does not handle updates.

Note: actual file I/O for Inserts and Updates happens in TakeCompleted. See How Not To Ignore The Form Template.

But, I have my starting point. PrimeUpdate,Parent call should be the point at which a user confirms a delete and, when there is client-side autonumber key, where a placeholder records is inserted.

PrimeUpdate

The first place to look is in the browse. This is the logical starting point because I/O preparation has been moved to the browse to support Edit in Place.

Indeed, checking the embed tree in any browse shows a PrimeUpdate embed. Starting with BrowseCUSTOMER in Tutor.APP, I copied the procedure to BrowseCUSTOMERsequence and put STOPs before and after the Parent call in PrimeUpdate.

In my revised Tutor.EXE, on the main menu select: Browse | Sequence of Events. Try inserting a new record or deleting an existing one. Neither of the STOPs appears.

Okay, maybe the PrimeUpdate method in the browse is only triggered if Edit in Place is active. In the demo app, BrowseCUSTOMER_EIP is an EIP-enabled browse built on the original BrowseCUSTOMER.

From the main menu, select Browse | EIP Browse and try to insert or delete a record. Still no STOP message.

Now I'm really mystified. But, to satisfy myself that the update procedure was actually being called, I placed STOPs before and after the update procedure is called. I've commented them out in the supplied EXE. But, feel free to uncomment them for your own testing. The update form *is* called; these STOPs *do* appear.

It turns out that Edit in Place has its *own* PrimeUpdate embed. I wasn't really aware of that at the time I created the demo application (I don't use EIP that often).

What's a poor programmer to do? Try the form, of course!

Form template procedures also have a PrimeUpdate method and the attendant embeds. The procedure BrowseCUSTOMERsequence2 is called from the main menu Browse | Sequence with form. If you try to insert or delete a record, halleluiah, the STOPs do appear.

So, for the moment ignoring Edit in Place, the update form's PrimeUpdate method is where deletes (and autonumber inserts) occur. And, theoretically, PrimeUpdate, Before Parent call is before a record is actually touched and, in that embed, I can stop a user from deleting a record. PrimeUpdate, After Parent call is after the user has confirmed or aborted the delete, perfect for post processing.

If the user confirmed the Delete, After Parent call:

```
If Self.Request = DeleteRecord and SELF.Response = RequestCompleted
   ! cascade deletes or do whatever else you want to do
```

```
End
```

(RequestCancelled, the default value in this embed indicates the user did not confirm the delete) ought to work.

And, in practice ...

That's the theory. Does it work in practice?

Well, I don't suspect I'd be writing this if it was a total bust, would I?

> **Note:** The demo for this section is called Tutor2.APP. The EXE is provided locally linked. For this app, I removed the "Cascade on Delete" constraint in Tutor.DCT.

In Tutor2.APP, UpdateCUSTOMER2 is a form procedure in which I use the form to bypass deletes. In Tutor2.EXE, from the main menu, select Browse | Use Browse to Prevent Deletes. You should get a message saying you can't:

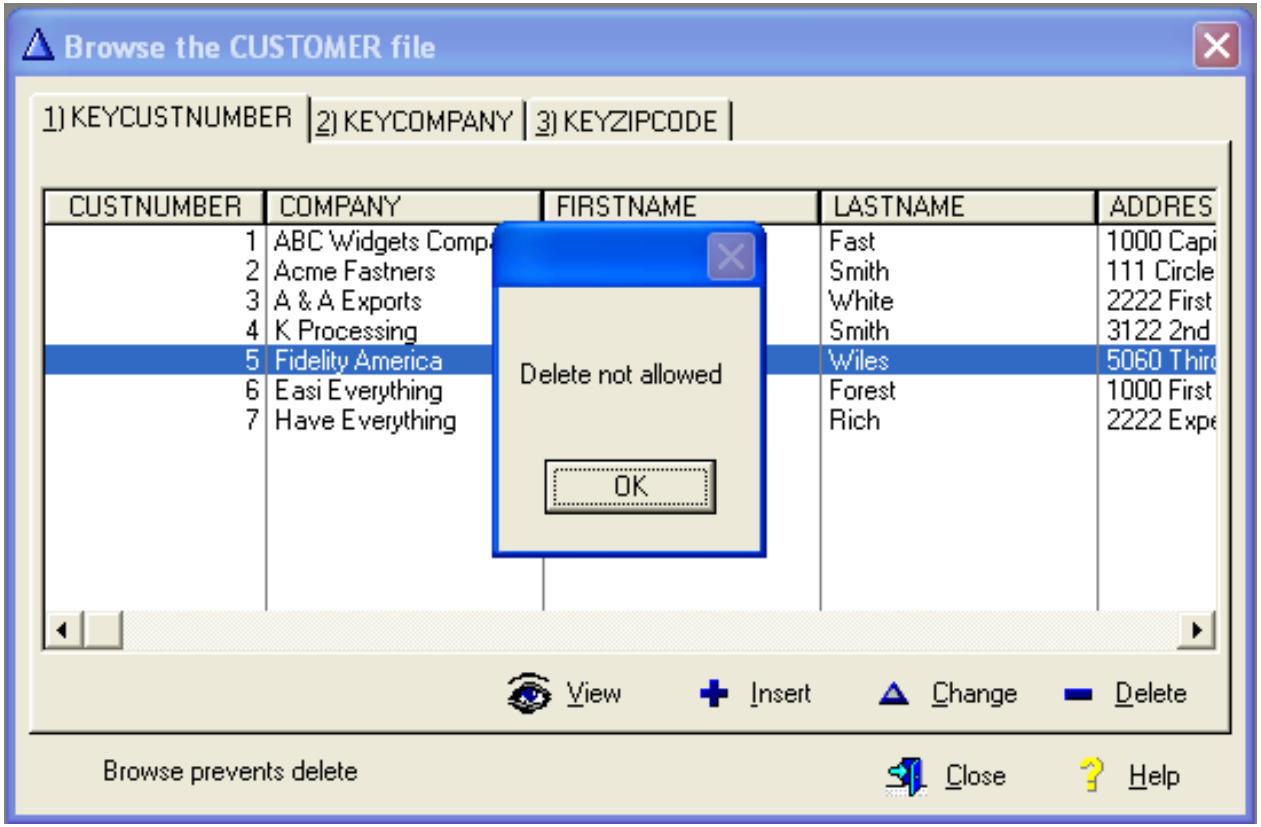


Figure 1. Delete not allowed message

In the Form's PrimeUpdate method, priority 4500, is this code:

```
If ThisWindow.Request = DeleteRecord
  Message('Delete Not Allowed -- Setting Zip to 99999','Customer ' |
            & CUS:CUSTNUMBER)
  CUS:ZIPCODE = 99999
  Access:Customer.Update
  Return Level:Fatal
!  Return 0             ! form still called
End
```

In addition to bypassing deletes, I also set the postal code to 99999 (I didn't have a flag field).

There is more code in the app.

If I want to know if there are records in Orders, a child of Customers:

```
Count = 0
Access:Orders.UseFile
Clear(ORD:Record)
ORD:CUSTNUMBER = CUS:CUSTNUMBER
Set(ORD:KEYCUSTNUMBER,ORD:KEYCUSTNUMBER)
Loop Until Access:Orders.Next()
  If ORD:CUSTNUMBER <> CUS:CUSTNUMBER
    Break
  End
  Count += 1
End
```

Code like this could be used to check whether to present a message to the end user that the customer has outstanding orders and cannot be deleted.

In a real app, before deleting a Customer, I check whether or not the customer has an Accounts Receivable balance. If there is a balance, I offer the end user the option to write off this balance:

```
If CustomerBalance <> 0
  CASE MESSAGE('This customer has an A/R balance of '  |↵
       & Clip(Left(Format(CustomerBalance,@n-13.2))) & '.', |
      'A/R Balance', ICON:Hand, |
       '&OK|Override', 1, 0)
  OF 1  ! Name: &OK  (Default)
```

```
    Cycle
  OF 2  ! Name: Override
    Override = 1
  END !CASE
End
```

(there is actually another check in the actual code). Only if the user selects "Override" is the delete performed, otherwise it is bypassed.

In another embed, this code:

```
If ThisWindow.Request = DeleteRecord
  Message('Delete Not Allowed. Window will be closed.<13,10> ' & |
   'But there are ' &  Clip(Count) & ' orders that could be deleted here.')
  Post(Event:CloseWindow)       ! Message and form still called
End
```

simply disallows deletes. Note that

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

and

```
Return  0
```

(I have left my original code in the app) fall through and the standard "Are you sure?" window appears. Only Return Level:Fatal keeps that message from appearing.

All of this code is available for you to comment in/out, to play with and see how each performs.

Barring deletes

In the real world, when all I want to do is prevent a user from deleting records, I would use this kind of code:

```
If <condition>
  ?BRWx.DeleteControl = 0
  ?Delete{Prop:Disable} = True
Else
  ?BRWx.DeleteControl = ?Delete
  ?Delete{Prop:Disable} = False
End
```

in TakeNewSelection or in UpdateWindow (see Controlling Controls – code like this is easily adapted to conditionally turning on or off inserts and updates also).

I can also ask if a user wants to delete or flag a record in the browse (the write-off code, above, is from the delete button on a browse – note how I CYCLE out instead of RETURNing).

In other words, the point of this exercise is not to determine the best place to bypass deletes or to offer the end user a "delete or flag" choice. The point is to verify that the Form's PrimeUpdate method *is* the place where delete in fact happens.

Post Processing

This is the real reason for the exercise. Suppose I need to do something after the end user OKs a delete. Perhaps I need to write the A/R write-off mentioned above. Or, I might want to do my own RI deletes (especially for non-SQL databases as discussed in Where Delete Occurs, Part 1),

In Tutor2.EXE, from the main menu, select Browse | Post Process RI. You will get a message telling you that you are about to delete a customer and you will see the customer number (I just wanted to be certain I had the right record, otherwise this message is unnecessary).

Then you will get the standard "Are you sure?" message. If you decline the delete, you are returned to the browse, exactly as expected. But, if you okay the delete, you will get a confirmation that the delete occurred and how many orders need to be deleted:

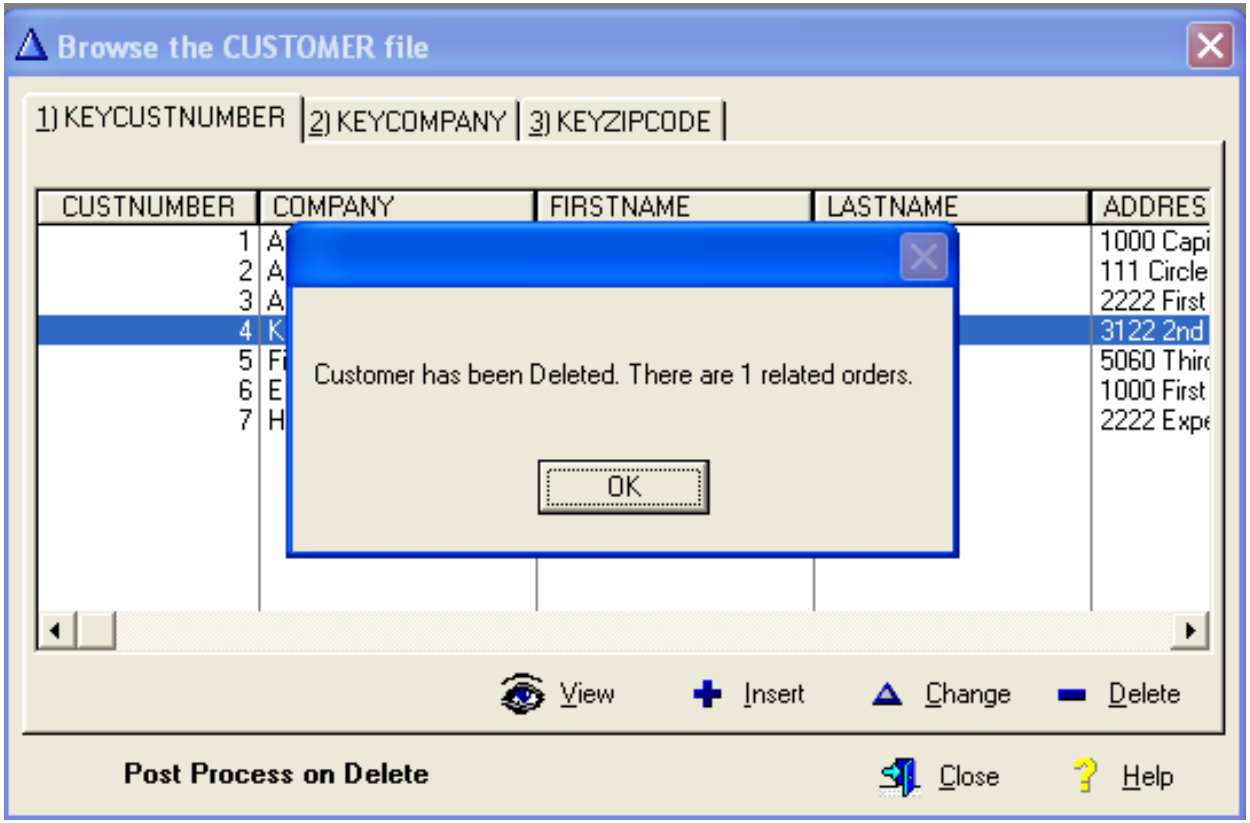


Figure 2. Post Processing RI

(in the demo app, I don't actually delete the related records, I just count them – in a real app, I would have cascaded deletes as shown in part 1). You will find the code in PrimeUpdate, After Parent call, exactly as expected.

Summary

At least when an update form is used, delete happens in the Form's PrimeUpdate method. By judicious use of the before and after parent call embeds, I can gain significant flexibility. I can do my own RI or other post-delete processing. I can flag records "inactive" or actually delete them. I can offer the user a choice.

The only thing missing is complete generality. PrimeUpdate works perfectly for fine tuning program behavior when deleting records and, perhaps, that is enough for you. Tweaking before and after Inserts and Updates, however, cannot be handled here.

Next time ....

Download the source

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

Reader Comments

Posted on Friday, March 26, 2010 by Phil Will

Always enjoy your articles :). Another place to capture the delete is in ThisWindow.Run in the browse procedure before calling the update procedure. This is a useful way to code before and after various procedures are called. For deletes, the example below bypasses the update procedure altogether. Each browse has a procedure number contained in the "AskProcedure" property that is used by ThisWindow.run.

```
ThisWindow.Run PROCEDURE(USHORT Number,BYTE Request)

ReturnValue         BYTE,AUTO

! Start of "WindowManager Method Data Section"
! [Priority 5000]

! End of "WindowManager Method Data Section"
  CODE
  ! Start of "WindowManager Method Executable Code Section"
  ! [Priority 2500]
  IF number=brw1:AskProcedure and Request=DeleteRecord
    ! do your delete code.
    RETURN RequestCompleted
  END
```

```
! Parent Call
ReturnValue = PARENT.Run(Number,Request)
! [Priority 6000]

IF SELF.Request = ViewRecord
  ReturnValue = RequestCancelled                    ! Always return RequestCancelled if the form was opened in ViewRecord mode
ELSE
  GlobalRequest = Request
  UpdateCUSTOMER
  ReturnValue = GlobalResponse
END
```

Posted on Friday, March 26, 2010 by Steven Parker

Thanks, Phil, for the kind words.

Patience. There are more articles to come ....

Add a comment

Clarion Magazine

Where Delete Occurs, Part 3

by Steven Parker

Published 2010-03-29

It doesn't happen often, but sometimes I need to customize standard, template generate CRUD operations. Most frequently, I want to cascade RI deletes myself rather than accept the default template generated RI. "Most frequently," but there are other customizations I need/want/am requested to provide.

> Note: The demo for this article is Tutor2.APP. The EXE is provided locally linked. For this app, I removed the "Cascade on Delete" constraint in Tutor.DCT.

In Part 1 of this series, I discussed alternate ways to manually cascade deletes. I need to take control of this process, especially with SQL, because I may not have access to the database to create a stored procedure or trigger. In this circumstance, a trigger in the dictionary works wonders.

With TPS databases, the problem I find is large numbers of child records. When there are large child sets to delete, the record by record processing (check abfile.clw for RelationManger.DeleteSecondary, where standard RI code is generated) can be seriously slow. In those kinds of situations, I discovered that I could START another procedure, pass it the unique record identifier

```
Start(DeleteCustomer,,CUS:CustomerNumber)
```

and do my deletes on a separate thread (though there is a temporary out-of-sync condition between the parent and child files; the parent will be deleted before the STARTed delete procedures complete). Performance was quite acceptable.

The question to answer is "Where is the appropriate place for this code?"

In Part 2 of this series, I showed that the update form's PrimeUpdate, After Parent call method is after the user has accepted the delete (answered "Yes" to the standard "Are you sure you want to delete the highlighted record?" prompt):

```
If Self.Request = DeleteRecord and  SELF.Response = RequestCompleted
  Start(DeleteCustomer,,CUS:CustomerNumber)
End
```

PrimeUpdate, Before Parent call is an appropriate place to ask whether the user would rather flag the record "inactive" instead of deleting it. I showed how to do that while bypassing the standard "Are you sure?" prompt. It is also where I can bypass deleting entirely.

Any tweak I might want to apply to the delete process, I can, in a fairly straightforward manner, in the update form's PrimeUpdate method. But Inserts and Updates are not tweakable in PrimeUpdate.

What I really need is a more general way of modifying the standard template handling of CRUD.

Forward to the past

In days of old, when knights were ... and there was simply no way anyone would ever, conceivably need more than 640 KB of memory and CPD (Clarion Professional Developer) was a perennial PC Magazine Editor's Choice ... CPD featured two 20 character long entry fields on a Table's (browse) Screen Properties worksheet.

As I recall, one was called (something like) "Edit Procedure Before Update" and the other, "Edit Procedure After Update" the update form. (Actually, the field was 256 characters but you could only see 20 of them. These fields were intended to receive procedure calls but I, like most Clarioneers in those days, wrote code directly in them.)

Later, Clarion Database Developer, introducing the embed system, had embeds labeled "Prior to Update Procedure" and "After Update Procedure."

Embeds before and after the update procedure call are just what I need. Before the update procedure, I can test GlobalRequest or the local variable used to prime it, as Clarion continues to use the request/response model. From the help:

Local Variables:	ThisWindow.Request (or SELF.Request), ThisWindow.Response (or SELF.Response), ThisWindow.OriginalRequest (or SELF.OriginalRequest). ThisWindow.Request (or SELF.Request) and ThisWindow.OriginalRequest (or SELF.OriginalRequest) are assigned value immediately after the procedure begins.

So, before calling the update procedure, something like:

```
Case ThisWindow.Request
  Of InsertRecord
    ! do pre-insert stuff
  Of DeleteRecord
    ! do pre-delete stuff
  Of ChangeRecord
    ! do pre-update stuff
End
```

should tell me what is about to happen. I can do whatever pre-processing I might want. And, after the update procedure returns:

```
If ThisWindow.Response =  RequestCompleted
  Case ThisWindow.Request
    Of InsertRecord
      ! do post-insert stuff
    Of DeleteRecord
      ! do post-delete stuff
    Of ChangeRecord
      ! do post-update stuff
  End
Else
  ! user did not insert, delete or update
End
```

should allow me to do any post processing I might want.

Unfortunately, examining the embed tree reveals nothing like what I want.

But wait!

Clarion now has a source view, a view in which I can see all the code that could be generated. I should be able to use this to find where my update procedure is called. Once I locate my update procedure, I can easily see the embeds immediately before and after my update procedure.

```
ThisWindow.Run PROCEDURE(USHORT Number,BYTE Request)

ReturnValue          BYTE,AUTO

! Start of "WindowManager Method Data Section"
! [Priority 5000]

! End of "WindowManager Method Data Section"
  CODE
  ! Start of "WindowManager Method Executable Code Section"
  ! [Priority 2500]

  ! Parent Call
  ReturnValue = PARENT.Run(Number,Request)
  ! [Priority 6000]
    Before update procedure
  IF SELF.Request = ViewRecord
    ReturnValue = RequestCancelled
  ELSE
    GlobalRequest = Request
    UpdateCUSTOMER21
    ReturnValue = GlobalResponse
  END
  ! [Priority 8500]
    After update procedure
  ! End of "WindowManager Method Executable Code Section"
  RETURN ReturnValue
```

Figure 1. Where the update procedure is called

And, voilà, there it is! (Subtle, huh?) The two embeds I need are ThisWindow.Run, priority 2500 or 6000 (the parent call, in abwindow.clw, contains only Return RequestCancelled so I don't think there's really much difference between these embeds) and ThisWindow.Run, priority 8500. However, there are two ThisWindow.Run methods:

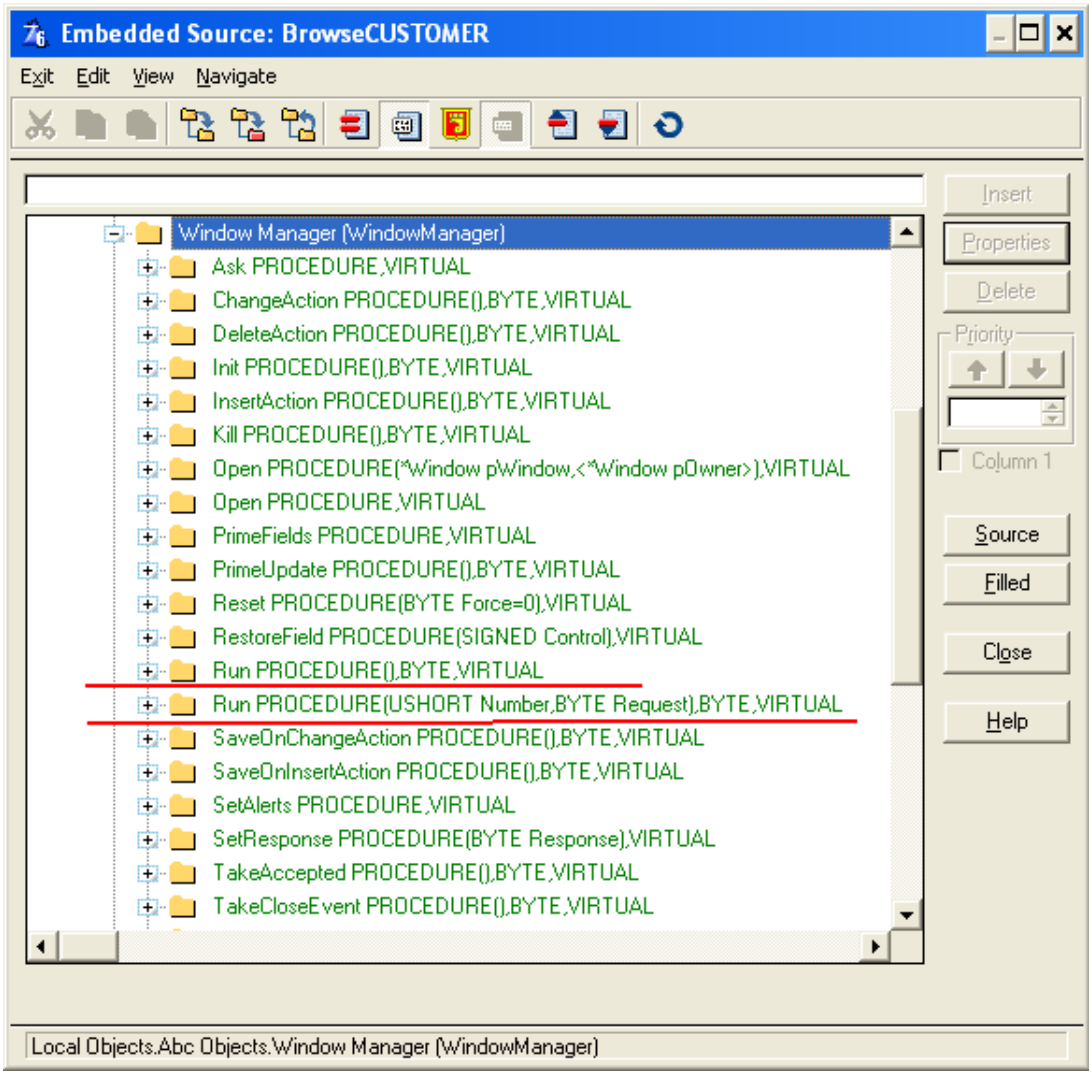


Figure 2. Embed tree

The Run method containing the update procedure call is the second, with the prototype:

```
ThisWindow.Run PROCEDURE(USHORT Number,BYTE  Request)
```

And, because Request is a parameter of the method, my code, above, doesn't need the object name (ThisWindow or Self).

Test Cases

Case 1: Can I use ThisWindow.Run to prevent a user from deleting a record?

In the demo app, from the main menu, select Browse | Use Browse to Prevent Deletes. Try to delete a record. You will get a message that I inserted and the record will not be deleted. The procedure in Tutor2.APP is BrowseCUSTOMERsequence2.

```
ThisWindow.Run PROCEDURE(USHORT Number,BYTE Request)

ReturnValue          BYTE,AUTO

! Start of "WindowManager Method Data Section"
! [Priority 5000]

! End of "WindowManager Method Data Section"
  CODE
  ! Start of "WindowManager Method Executable Code Section"
  ! [Priority 5000]


  ! [Priority 5000]
  If Request = DeleteRecord
    Message('Delete not allowed')
    Return RequestCancelled
  End
  ! Parent Call
  ReturnValue = PARENT.Run(Number,Request)
  ! [Priority 6000]

  IF SELF.Request = ViewRecord
    ReturnValue = RequestCancelled
  ELSE
    GlobalRequest = Request
    UpdateCUSTOMERsequence
    ReturnValue = GlobalResponse
  END
  ! [Priority 8500]

  ! End of "WindowManager Method Executable Code Section"
  RETURN ReturnValue
```

Figure 3. bypassing delete, in the browse

Case 2: Can I use ThisWindow.Run to flag a record instead of deleting it?

From the main menu, select "Set 'Inactive' instead of Deleting" (the procedure is BrowseCUSTOMER2). You will be asked whether you want to flag or delete the record. Depending on what you choose, you will either be returned to the browse (the postal code will be changed to 99999 – I didn't add a flag field to the dictionary) or get the standard "Are you sure you want to delete?" prompt.

```
ThisWindow.Run PROCEDURE(USHORT Number,BYTE Request)

ReturnValue          BYTE,AUTO

! Start of "WindowManager Method Data Section"
! [Priority 5000]

! End of "WindowManager Method Data Section"
  CODE
  ! Start of "WindowManager Method Executable Code Section"
  ! [Priority 2500]

  ! Parent Call
  ReturnValue = PARENT.Run(Number,Request)
  ! [Priority 6000]
  If Request = DeleteRecord
    If Message('Would you rather set this customer to "inactive" status ' &
        'instead of deleting?','Confirm',ICON:Question,Button:Yes+Button:No
      Access:Customer.Fetch(CUS:KEYCUSTNUMBER)   ! ensure record is retriev
      CUS:ZIPCODE = 99999
      Access:Customer.Update
      Return RequestCompleted
    End
  End
  IF SELF.Request = ViewRecord
    ReturnValue = RequestCancelled                          ! Always return
  ELSE
    GlobalRequest = Request
    UpdateCUSTOMER
    ReturnValue = GlobalResponse
  END
  ! [Priority 8500]

  ! End of "WindowManager Method Executable Code Section"
  RETURN ReturnValue
```

Figure 4. Ask "flag or delete" in the browse

Case 3: Can I manually cascade RI deletes in ThisWindow.Run?

Taken from a production app, this picture shows that I can:

```
ThisWindow.Run PROCEDURE(USHORT Number,BYTE Request)

ReturnValue          BYTE,AUTO

! Start of "WindowManager Method Data Section"
! [Priority 5000]
SAV:DEV:DevID   Long
! End of "WindowManager Method Data Section"
  CODE
  ! Start of "WindowManager Method Executable Code Section"
  ! [Priority 50]

  ! Parent Call
  ReturnValue = PARENT.Run(Number,Request)
  ! [Priority 5500]
  If Request = DeleteRecord
    SAV:DEV:DevID = DEV:DevID
  End
  IF SELF.Request = ViewRecord
    ReturnValue = RequestCancelled                          ! Always return
  ELSE
    GlobalRequest = Request
    UpdateDevice
    ReturnValue = GlobalResponse
  END
  ! [Priority 7500]
  If Request = DeleteRecord and GlobalResponse = RequestCompleted
    Rules{Prop:SQL} = 'Delete from Rules where Rules.DeviceID = ' & SAV:DEV
    Rules{Prop:SQL} = 'Delete from Pulses where Pulses.DevID = ' & SAV:DEV:
  End
  ! End of "WindowManager Method Executable Code Section"
  RETURN ReturnValue
```

Figure 5. Manual RI in ThisWindow.Run

In the screen shot, you will note that I save the unique record identifier when deleting

```
If Request = DeleteRecord
  SAV:DEV:DevID = DEV:DevID
End
```

and use the local variable in my post processing. This is just a bit of paranoia on my part, ensuring that I have the correct value even if DEV:DevID gets cleared somewhere.

Summary

Clarion Database Developer introduced "set and forget" RI. The templates picked up dictionary options and generated the RI for me. Because I had to do all this manually in CPD (using those small "Edit Procedure" prompts), this was a big step forward.

However, there are times when I can do it better than the templates. There are times I need to significantly change the way standard operations behave.

For deletes, the Form template's PrimeUpdate method works quite nicely. However, a single point exists where I can do just about anything I want for any file operation, ThisWindow.Run. I don't need it often. But when I do, there's just no substitute.

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Where ... Stuff Occurs: Deriving the FileManager

by Steven Parker

Published 2010-03-30

I sometimes need to intercept CRUD operations. Sometimes I want to stop the operation from happening. Sometimes I want to offer the user options. Sometimes I want to do something additional when the operation completes.

This led me to seek where these operations occur. As the previous articles in this series testify, I found there were several options available to me.

However, with one exception (client side triggers on delete), all involved embedded code in browses and forms. This necessitates embedding code in every browse or form – depending on the particular operation and intercept technique – where the action I want to customize occurs. Additionally, I run the very real risk of forgetting to handle an instance of a file operation. In addition, the very idea of embedding code has recently come under attack by the editor of the world's most popular Clarion publication, but he shall remain nameless.

> [Nameless editor: *Actually what I attacked was the wholesale embedding of business logic in embed points, rather than (where possible) placing that logic in testable, reusable classes.*]

"He who may not be named" thought "It might be worth investigating/explaining the difference between one-off embed code changes and deriving the FileManager or RelationManager."

If it is possible to use the FileManager or RelationManager, I would have a single point to insert my code. I could be much more comfortable that I haven't omitted a place with an operation in which I was interested. In other words, using the FileManager or RelationManager would be global, totally set and forget.

So, I was thinking ...

There are a number of global embeds that appear to be related to CRUD operations. These include field validation embeds for every file and field known to the app:

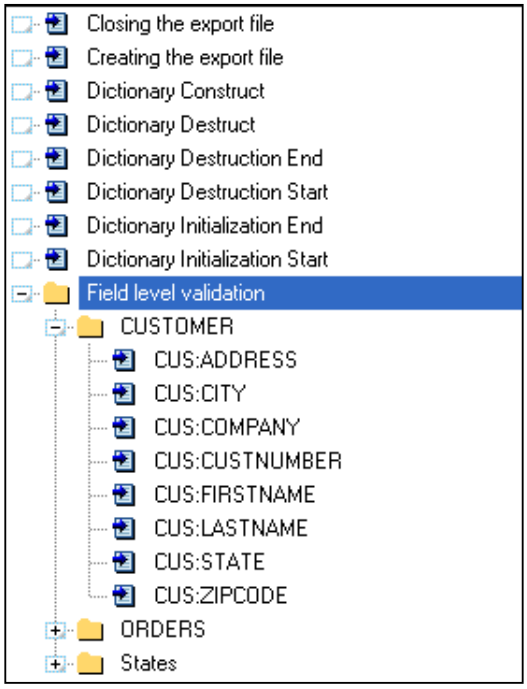


Figure 1. Global Field Level Validation embed

Sure enough, there are all manner of global FileManager embeds, for each FileManager, surrounding virtually every step of every Insert, Update or Delete operation:

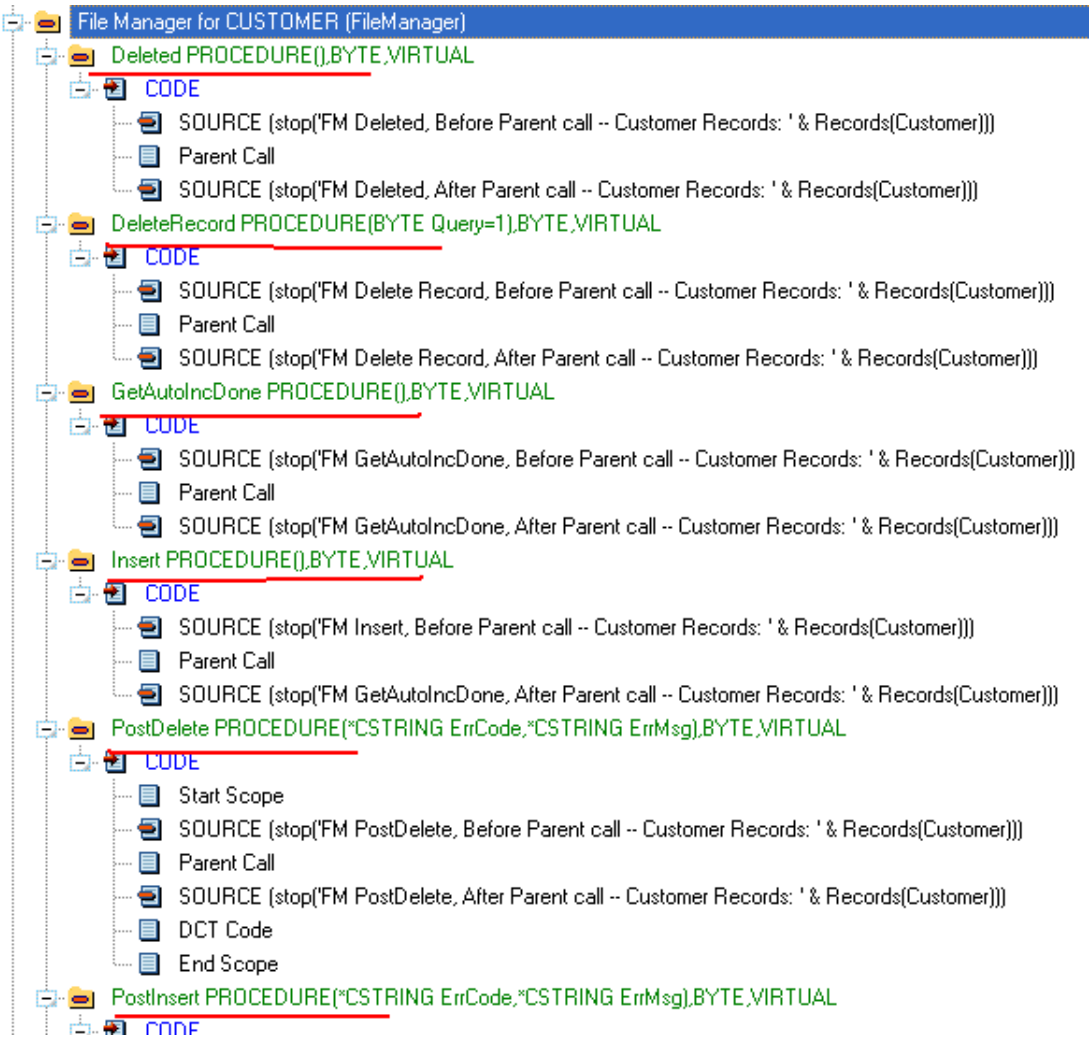


Figure 2: Global FileManager embed tree

So, in those cases where I am entirely certain I *always* want to do something, these FileManager embeds appear to be just the ticket.

Where does <*anything*> occur?

The demo app is downloadable at the end of this article. It is based on the Tutor.APP in the Examples folder and is a 6.3.9056 app. A locally compiled version is also supplied.

I put STOPs in every embed that had anything to do with CRUD ops (Customer file), both before and after the Parent call. Checking both before and after is necessary because any of the method's parent call could call one of the other methods.

At each STOP, I display the record count for the file. This makes it easy to tell when the file has changed.

Inserts (with autonumber key)

For a file with an autonumber key, the sequence of events is as follows:

Event	Record count
Initial record count	*7*
PrimeAutoInc, Before Parent call	7
PreInsert, Before Parent call	7
PreInsert, After Parent call	7
PostInsert, Before Parent call	8
PostInsert, After Parent call	8
PrimeAutoInc, After Parent call	8
Form appears	
Insert, Before Parent call (if Insert is canceled, DeleteRecord is called)	8
PreUpdate, Before Parent call	8
PreUpdate, After Parent call	8
PostUpdate, After Parent call	8
GetAutoIncDone, After Parent call	8

Table 1

Note where the record count goes from the initial seven to eight between PreInsert, After Parent call and PostInsert, Before Parent call. This implies that the autonumber placeholder record is added at PreInsert, After Parent call. Or, more correctly, PreInsert, After Parent call is the last embed before the placeholder is added.

In the demo app (ignore, for the moment, some messages I added for other purposes), select Browse | Browse the CUSTOMER File.

By the way, this sequence holds for Edit in Place, which I had trouble tracing in the previous articles. In the demo app, Browse | Browse Customers EIP will follow the sequence above.

Inserts (without autonumber key)

If the table does not have a client side autonumber key, I expect the sequence of events to be somewhat abbreviated compared to the sequence shown above. In the demo app, I embedded code in all the same embeds for a file that does not have an autonumber key (the States file): Browse | Browse the States file shows the following:

Event	Record count

Initial record count	*4*
Form appears	
Insert, Before Parent call	4
PreInsert, Before Parent call	4
PreInsert, After Parent call	4
PostInsert, Before Parent call	5
PostInsert, After Parent call	5

Table 2

The record is actually added at PreInsert, After Parent call. At least this is the last embed before the new record is written to file.

This embed is the same as a file that does have an autonumber key. It's nice to see consistency between the with and without autonumber key scenarios.

Updates

The sequence of events when changing an existing record is:

Event
Form appears
PreUpdate, Before Parent call
PreUpdate, Before Parent call
PostUpdate, After Parent call
PostUpdate, Before Parent call

Table 3

In the demo app, I have embedded STOPs in the Customer Browse to track updates. If you open a customer record in update mode and press the OK button, making no actual changes to the data, none of the embeds are triggered. You'll see my messages only if you actually change something. That's pretty cool and exactly what I would expect.

Deletes

Testing deletes, again from the customer browse, I also want to track the customer number. If the customer number is not current and correct at any point, I will have a problem with any code I embed that uses range limits or filters on the customer number:

Event	**Record Count**	**Customer Nbr**
Customer selected for delete	*8*	*8*
Standard "Ask" appears		
DeleteRecord, Before Parent call	8	8
PreDelete, Before Parent call	8	8
PreDelete After Parent call	8	8
PostDelete, Before Parent call	7	8
PostDelete, After Parent call	7	8
DeleteRecord, After Parent call	7	8

Table 4

Here, it looks like the last embed before the physical delete is PreDelete, After Parent call.

Customizing Operations

The FileManager provides a plethora of embeds that appear to allow me to apply code at an almost atomic level to modify how those operations occur.

In fact, on the surface it looks like there are too many embeds available. It looks like I will have a problem finding the right embed, there are so many.

Unfortunately, it just isn't so. Very few of the things I may want to do can be effected in FileManager embeds.

For example, consider stopping a delete from happening or writing an "inactive" flag to a record instead of deleting it or asking the user whether to flag or delete. If I want to do any of these three things, my code needs to execute before the standard "Are you sure?" message comes up.

But, look at the chart of events for delete, in Table 4 above. There is no FileManager embed available before the "ask." You can satisfy yourself by checking the demo app to see if you can find an embed I missed.

By itself, this could be enough for me to reject using FileManager embeds for "set and forget" code.

> [Nameless Editorial Musing: This would probably be a lot easier if Clarion had a true layered architecture. But to some extent there are at least two layers here, the data access layer (Relation/FileManager) and the UI layer (the "ask"). So you really have to decide whether you want to prevent a delete at one or both levels, both usually making the most sense. You want to prevent it at the data layer, but you want to either prevent the user from asking for a delete in the first place, or provide some useful information if they try and it can't be done.]

However, that would be premature. If, for example, I want to log the delete, the PostDelete, Before Parent call embed really is perfect. This embed is called immediately after the physical delete of the record. I presume that if I call

```
Relate:Customer.Delete(0)
```

because I am calling a RelationManager method and the RelationManager calls the FileManager's delete method, my logging code *will* be called for each record. Further, if the user decides not to delete the record, this embed – as I would expect – is not called.

But, for bypassing a delete or for flagging instead of deleting, overriding the FileManager won't do.

Similarly, if I want to stop a user from changing a record, there is no embed before the form appears (see Table 3). While I did not test this, if my interest is in logging, my guess is that the embeds shown in Table 3 above will work as expected. That means that they will also work for a cascaded change (from the dictionary) or for

```
Relate:Customer.Update
```

Inserts

Inserts are a very different story. Comparing the embed flow for files with and files without autonumber keys, it is easy to see the two sequences have little in common.

For post-processing, both share the Insert, Before (and After) Parent call embed. So, if I need to log inserts, using one of the Insert embeds means I don't have to stop and check if there is an autonumber key. That's to the good.

If I want to do something before the user inserts a record, however, the difference is significant. Without an autonumber key (or using Edit in Place with one), there is no FileManager embed before the form displays (see Table 2, above). But, if there is an autonumber key, there are a number of embeds available (see Table 1).

In the demo app, for the customer file (autonumber key), I tried to circumvent an add at PreInsert, Before Parent call. I put a message and Return RequestCancelled. It didn't work. The remaining insert methods were called.

Embedding the attempt to bypass the add in PrimeAutoInc, Before Parent call did work. So, if you want to see the code in PreInsert fail, you'll need to comment out the code in PrimeAutoInc. And I do recommend you go through the exercise.

Summary

The FileManager held great promise for acting "where delete occurs." After all, the FileManager is the object that manages the file.

The reality is that overriding the FileManager seems best suited for acting after a CRUD op. But, for acting before CRUD happens, to intercept a delete for example, overriding the FileManager doesn't work (and it is for this reason that I have not considered multi-DLL apps – though I would expect that overriding the FileManager in the data DLL would all that is required).

This was an interesting exercise. The results were really unexpected. If nothing else, should I need to implement logging, I now know the place to do it.

Download the source

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

Reader Comments

Add a comment