

**Reborn Free****CLARION**  
*online***Validating Credit Card Numbers**

Credit card processing is a common requirement in application development. And before you submit a credit card number to your bank for processing, it helps to make sure it's a well-formed number. Abe Jimenez explains the code you need to do the job.

*Posted Tuesday, March 02, 2004*

**How To Stop Trashing The Template Registry**


How many times have you run a second instance of Clarion, without turning on multi-user development, only to have your registry trashed? In this article Danie de Beer describes a utility by Mark Goldberg that makes trashed registries a thing of the past.

*Posted Tuesday, March 02, 2004*

**Register For etc Now!**

You can now register online for the East Tennessee Clarion Conference! The conference fee is exactly the same as two years ago: \$469 covers all general sessions for the three core days of the conference: Wednesday through Friday. Also included is the reception Tuesday evening, breakfast and lunch on all three days, the Cajun Cookout, the closing dinner, and breaks

Articles: 

News: 

**News**

[Embedded Firebird](#)

[xInactivity 1.1](#)

[In-Memory Database Driver Information](#)

[xInactivity Available On ClarionShop.](#)

[Advanced Data Dictionary Architect 1.0.2](#)

[Icetips Previewer 2.0](#)

[Multilingual Text To Voice](#)

[Fenix ASP.NET Generator at ETC 2004](#)

[1st Logo Design Spring Specials](#)

[Clarion Third Party Profile Exchange Updated](#)

[Firebird Links And Tutorial](#)

[SealSoft xTipOfDay 2.0](#)

[ClarionForge Input Requested](#)

[CapeSoft Training At ETC 4](#)

[SB5 Beta Online Forum](#)

[Clarion And Armadillo HowTo](#)

[DCT2SQL For Interbase](#)

**SURVEY**

**Which is your favorite NOT unary operator?**

NOT  41.1%

~  58.9%

56 responses

[Previous Surveys](#)

**One Year Ago In CM**

[GPF Challenge Results](#)

[Mutexes: Serializing File Access](#)

[Potholes On The Road To Open Source Database Nirvana](#)

**Two Years Ago In CM**

during the sessions. Special rate for Edgewater hotel rooms. Guest meal plans and Saturday session by Bruce Johnson also available.

*Posted Wednesday, March 03, 2004*

### **If you emailed ClarionMag and haven't had a reply yet...**

Vacation's over! The Clarion Magazine office is again open, and all books ordered during our break have now been put in the print queue. Our spam filtering software also picked up about 10,000 spam emails during this time, and because of the quantity we deleted most of it sight unseen. If you haven't yet received a response to an email you sent in the last 10 days, please send it again.

*Dave Harms, Publisher*

*Posted Monday, March 15, 2004*

### **Weekly PDF for March 1-6, 2004**

All articles for March 1-6, 2004 in PDF format.

*Posted Tuesday, March 16, 2004*

### **A Class Wrapper for Brice Schagane's Menu Buttons**

Nik Johnson gets his copy of Tips & Techniques and quickly discovers a solution to his screen real estate problem: Brice Schagane's menu button. For easier re-use, Nik shows how to convert Brice's code into a class.

*Posted Friday, March 26, 2004*

### **Using Client-Side Triggers In**

[MAV Direct ODBC Rewrite](#)

[Gitano Buy One Get One Free](#)

[xDataBackup Manager Pro 1.7](#)

[Free Web Email Link Cloaking Utility](#)

[Bg XML Dico Video](#)

[Excel Charts 1.0](#)

[gReg Plus \\$99 Offer](#)

[Sticky Notes For C6](#)

[EasyVersion 2.02](#)

[File Manager 3 Beta 3.30](#)

[New Wallpaper Template](#)

[Encourager Software Web Site Redesign](#)

[Solace Software ReSort Pro & New Web Site Design](#)

[MAV Direct ODBC 0.04 And 0.07](#)

[Freeware Templates Available](#)

[Mail & Merge Manager v1.1](#)

[Outlook Products Available At ClarionShop](#)

[New Clarion Dictionary Utility](#)

[O'Reilly Launches New Windows DevCenter](#)

[Search the news archive](#)

[Baby Overrides Publishing Schedule!](#)

[Creating A SCADA Interface With Clarion \(Part 2\)](#)

[Secondary Forms \(Part 2\)](#)

### **Three Years Ago In CM**

[Clarion and the Internet: Publishing Static Data](#)

[Dynamic Filters: Applying The Theory](#)

[Clarion News - April 2001](#)

### **Four Years Ago In CM**

[The Clarion Challenge: Shoot Yourself In The Foot Results](#)

[Using Clarion With MySQL - Part 1](#)

[Doodling Bitmaps](#)

## **Clarion 6**

Client-side triggers are a new, very neat and useful feature that has been added to Clarion 6, both Professional and Enterprise versions. Tom Giles provides an introduction.

*Posted Friday, March 26, 2004*

## **Understanding Clarion Templates, Part 1**

Templates! Templates! Templates! That for many years has been the Clarion rallying cry (unless you prefer David Bayliss's "Don't Know, Don't Care!" from DevCon '97). And despite the advance of Clarion OOP technology, templates are still what Clarion does best. But what are the templates, really? David Harms begins this series with an overview of template types.

*Posted Friday, March 26, 2004*

## **Our Email Addresses Have Changed!**

Please note that due to excessive spam we have been forced to change all email addresses. Please see this page for the new addresses.

*Posted Friday, March 26, 2004*

## **David Harms To Speak at ETC 2004**

David Harms, Clarion Magazine's publisher, will be giving a presentation on XML at the East Tennessee Clarion Conference and Gathering (also known as ETC 2004). As David's presentation is immediately prior to the Cajun Cookout, attendees are requested to keep drooling to a minimum. Nothing

unnerves a presenter more than that hungry dog look.

*Posted Tuesday, March 30, 2004*

### **Icetips Bios All Online**

Clarion Magazine is pleased to announce that it is the new home of the Icetips bios. By special arrangement with Sue Pichotta, these profiles of Clarion developers will remain free access. All the Icetips bios are now available.

*Posted Wednesday, March 31, 2004*

### **Tips & Techniques Book 15% Off Sale Ends April 16**

The 15% off introductory special on the Tips & Techniques book ends April 16. Order now before the price goes up!

*Posted Wednesday, March 31, 2004*

### **Understanding Clarion Templates, Part 2**

In this second article in his template series, David Harms takes a closer look at the template language itself.

*Posted Wednesday, March 31, 2004*

### **System Tray Popup Windows**

Shortly after installing Outlook 2003 Jim Kane noticed small windows appearing in the tray area when a new email message came in. He found these very useful, so he set about creating the same kind of popup messages in Clarion.

*Posted Wednesday, March 31, 2004*

### **Compiled Reports From Report Writer**

As far as Henry Plotkin is concerned, the best thing about Clarion Report Writer is the ability to set up the finished page without having to adjust the position of each individual band. The next best thing about Report Writer is that report testing is much faster. It does not require a compile-link cycle. It does not require minimizing Clarion, starting the application and navigating to the report. Wouldn't it be nice to be able to design reports in Report Writer, and then just copy them into a Clarion report procedure? Actually, it can be done, and fairly easily.

*Posted Wednesday, March 31, 2004*

Looking for more? Check out the [site index](#), or [search the back issues](#).

This site now contains more than 700 articles and a total of over a million words of Clarion-related information.

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

## [Clarion Magazine](#)



# How To Stop Trashing The Template Registry

by **Danie de Beer**

Published 2004-03-02

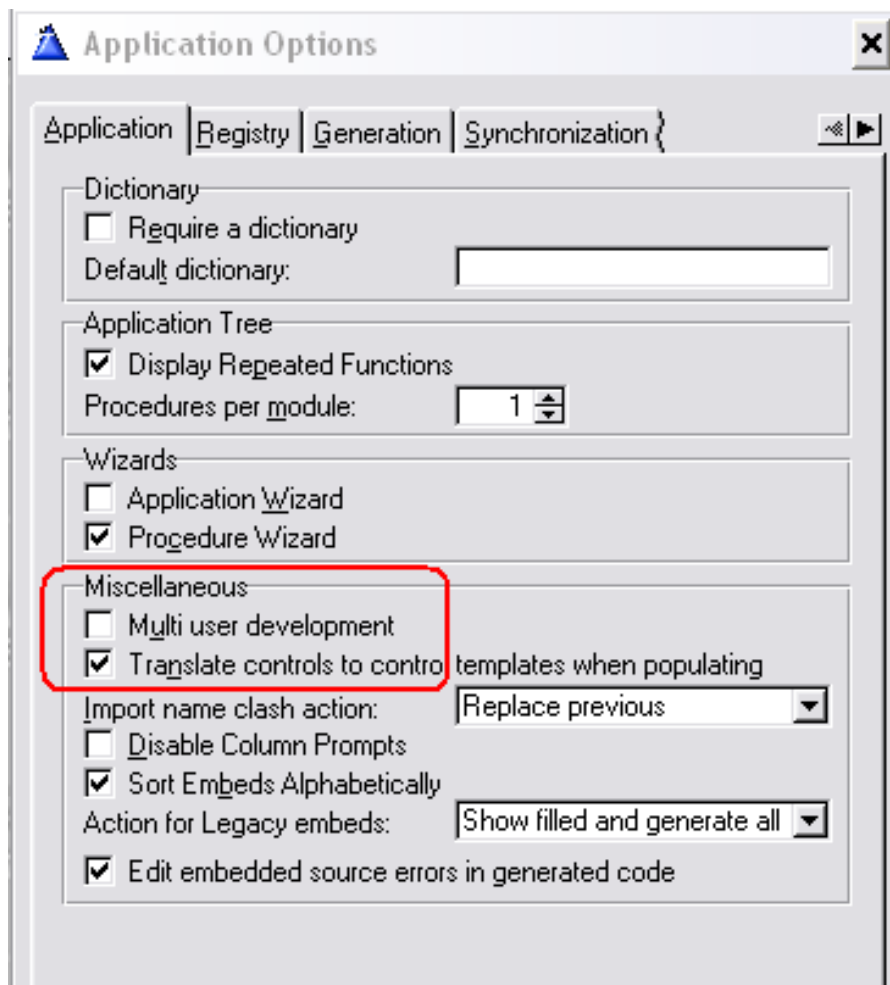
Source code is by Mark Goldberg, used with permission.

How many of you have done the same as I have a hundred times, you run Clarion, and then realize, Wow! Clarion is already running, and your Clarion IDE is *not* set for Multi-User Development. In a few seconds your Registry.TRF file is trashed, and you see the error message in Figure 1.



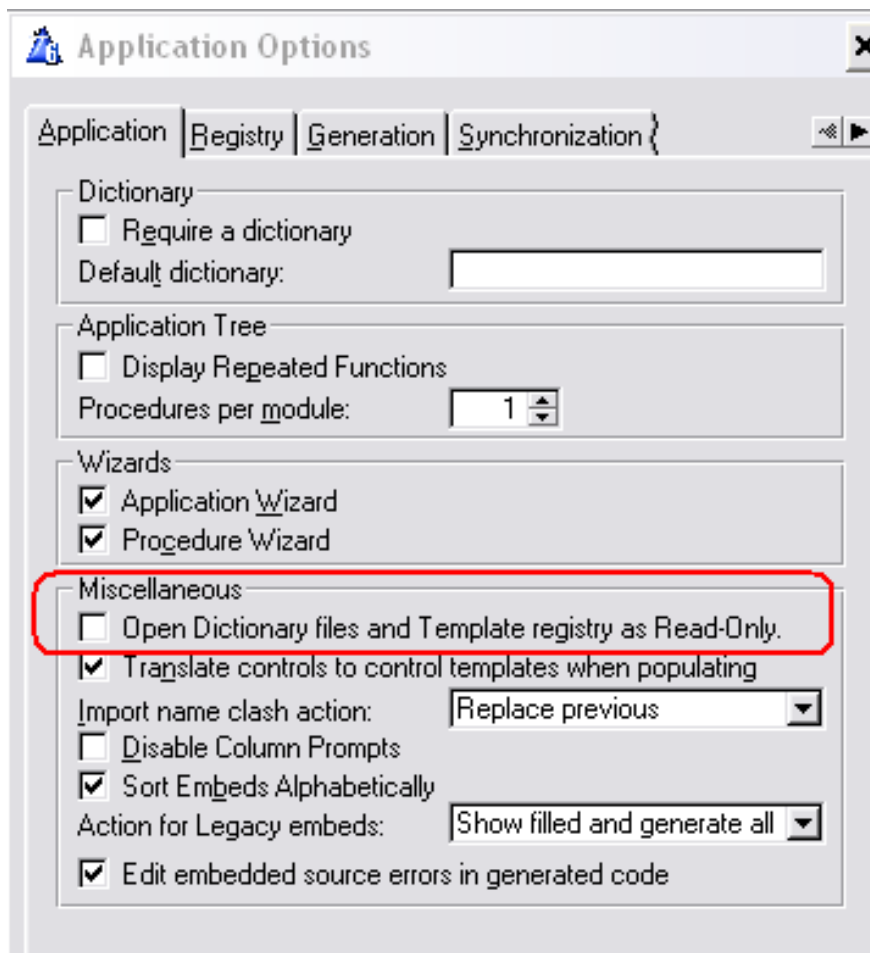
**Figure 1. A trashed registry**

I personally don't like switching the Multi-User Development setting on. Since this makes the registry read-only, every time I need to register a template, I need to switch it off and restart the Clarion IDE. In Clarion 5.5 you will find the Multi-User setting under the Setup, Application Option Menu, Miscellaneous Group



**Figure 2. Registry settings in Clarion 5.5**

In Clarion 6 the checkbox reads "Open Dictionary files and Template Registry as Read only."



**Figure 2. Registry settings in Clarion 6**

Some of us were discussing this issue on IRC (#cw-talk) one day, and following that discussion Mark Goldberg wrote this really handy utility for me in a couple of minutes. Instead of launching Clarion directly, you run this utility, which detects if Clarion is already running, and if it has been set to Multi User Development. If not, it informs you of your really horrible mistake, and in a split second prevents your registry from getting trashed. Otherwise the utility simply launches the Clarion IDE.

### **How it works**

First of all, this utility needs to know the INI settings of all the Clarion versions as listed in the Win.INI file.

In the BIN folder of each version of Clarion is an INI file that contains the MULTIUSER setting. So the first question becomes, where is the .INI file. The answer is, it's in **workdir** folder, listed in WINI.INI. Each version of Clarion has a different section name as you can see from the following



## example:

```
[Clarion 6.0 Enterprise Edition]
bin=BIN\
workdir=C:\Clarion6\BIN\
root=C:\Clarion6\
exename=C60EE
[Clarion 5.5 Enterprise Edition]
bin=BIN\
root=C:\C55\
exename=C55EE
workdir=C:\cla\C55\BIN\
```

The utility will also use **exename**, but I'll explain that later on.

The next step is to use a mutex to determine if the IDE is already running. The word mutex comes from the words MUTually EXclusive. Here's what the Clarion 6 help has to say on mutexes:

Mutexes are another kind of synchronization object. Their goal is the same as for critical sections: provide mutual exclusive access to some shared resource. For example, to prevent two threads from writing to shared memory (global data) at the same time, each thread waits for ownership of a mutex object before executing the code that accesses the shared resource.

There are 2 major differences between critical sections and mutexes:

Critical sections can only be used for synchronization of threads owned by the same process. Mutexes can be used for synchronization of threads that can belong to different processes. If a process creates a mutex with some name and another (or the same) process has created a mutex with that name already, the system does not create a new mutex. It returns another handle to the existing mutex instead and sets error code to 183 (ERROR\_ALREADY\_EXISTS). Unnamed mutexes are local for the process that created them. Such mutexes can only be used for synchronization of that process's threads.

The utility asks for a named mutex from Windows, and will only be able to get it if the mutex hasn't been "checked out" already, e.g. by another instance of the utility (the Mutex class was written prior to C6 being available; C6 has its own Mutex):

```
if Mutex_IDESingle.Init('CWIDE_Single' |
    & lcl:Argument.VerNum) or lcl:MultiUser
```

Since the name of the mutex includes the version number of the IDE, LaunchIDE can launch different versions of Clarion on the same machine at the same time.

## Data

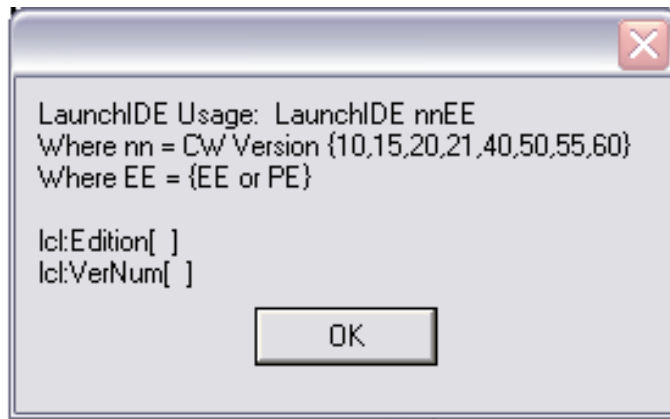
The utility will need a few variables, as follows:

```
lcl:WorkDir      String(260)      !Clarion working Directory
lcl:Ini          string(260)
lcl:Section     String(60)      !Ini Section
lcl:ExeName     String(260)     !Clarion Executable filename
Next, create a Group to store the Version number and Clarion Edition
lcl:Argument     Group,Pre()
lcl:vernum      String(2)       !Value like '55','60'
lcl:Edition     String (2)     !Values like 'EE','PE'
end
lcl:MultiUser   Byte          !Flag to check for Multi-User
                ! Development Switched on/off
```

You need to run the utility with a command parameter, e.g. LaunchIDE 60EE will launch Clarion 6.0, Enterprise Edition. Here's the code:

```
Lcl:Argument=command('1') !expecting a command line like: launchIDE
60EE
```

There's a small routine called ShowUsage that verifies that the command line parameters are correct. If there's a problem it displays a usage message, shown in Figure 4.



**Figure 4. The usage message**

Here's the code that determines which version of Clarion to run:

```

case lcl:Argument.VerNum
of ''
do ShowUsage
of '60'
case lcl:Argument.Edition
of 'EE'
lcl:Section = 'Clarion 6.0 Enterprise Edition'
lcl:Ini = 'c60ee.ini'
of 'PE'
lcl:Section = 'Clarion 6.0'
lcl:Ini = 'c60pe.ini'
else
do ShowUsage
end
of '55'
case lcl:Argument.Edition
of 'EE'
lcl:Section = 'Clarion 5.5 Enterprise Edition'
lcl:Ini = 'c55ee.ini'
of 'PE'
lcl:Section = 'Clarion 5.5'
lcl:Ini = 'c55pe.ini'
else
do ShowUsage
end
of '50'
case lcl:Argument.Edition
of 'EE'
lcl:Section = 'Clarion 5 Enterprise Edition'
lcl:Ini = 'c5ee.ini'
of 'PE'; lcl:Section = 'Clarion 5'
lcl:Ini = 'c5pe.ini'
else
do ShowUsage
end

```

```

!----- I believe that there was only one
!----- version prior to 5.0
of '40'
  lcl:Section = 'Clarion 4'
  lcl:INI = 'clarion4.ini'
of '21'
  lcl:Section = 'Clarion for Windows V2.1'
  lcl:INI = 'cw21.ini'
of '20'
  lcl:Section = 'Clarion for Windows V2.0'
  lcl:Ini = 'cw20.ini'
of '15'
  lcl:Section = 'Clarion for Windows V1.5'
  lcl:INI = 'cw15.ini'
of '10'
  lcl:Section = 'TopSpeed: Clarion for Windows'
  lcl:INI = 'cw.ini'
else
  do ShowUsage
end

```

**Now that the utility knows which Clarion version to Run and the INI file to use, it checks the Working Directory:**

```
lcl:WorkDir = clip( getINI(lcl:Section,'workdir') )
```

**Next, find out if the IDE is set to use Mutli-User Development. If MultiUser protections are off, then do *not* allow a second launch as it can trash the Registry.TRF:**

```

!MultiUser=off
lcl:MultiUser = choose( upper(getIni('Application','MultiUser',|
  '',clip( lcl:WorkDir ) & lcl:INI)) = 'ON')
!Policy: Will prevent the SIMPLE case, of MultiUser is Off,
! and all copies are being launched from a single machine

```

**Now it's time to actually "check out" the mutex from the OS. The Missing<2,4,8> is a default value for the INI. It's just a bogus value that is very unlikely to exist in the INI, which makes it possible to compare the lcl:Exename to the default value to determine if it was in fact missing**

```

if Mutex_IDESingle.Init('CWIDE_Single' & lcl:VerNum) |
  or lcl:MultiUser
  lcl:ExeName = getINI(lcl:Section,'exename','Missing<2,4,8>')
  if lcl:ExeName = 'Missing<2,4,8>'
    Message('Sorry, but that version of Clarion cannot ' |
      & 'be found in your WIN.INI','Launch IDE',ICON:Hand)
  
```

```

else
  lcl:ExeName = clip(lcl:WorkDir) & lcl:ExeName
  Run( lcl:ExeName, 1) !the ,1 means WAIT until completed
end
else
  Message('Clarion IDE is already running, and ' |
    & 'MultiUser Protections are NOT on.', 'Launch IDE', ICON:Hand)
end
!Message('LaunchIDE is closing|VerNum['& |
! lcl:VerNum &']|Edition['& lcl:Edition &']')
return
!Note: The mutex is automatically KILL'ed by the .Destruct

ShowUsage Routine
  Message('LaunchIDE Usage: LaunchIDE nnEE|Where nn = Clarion Version' |
    & ' {15,20,40,50,55,60}|Where EE = {{EE or PE}}|lcl:Edition['|
    & lcl:Edition &']|lcl:VerNum['& lcl:VerNum&']|')
return

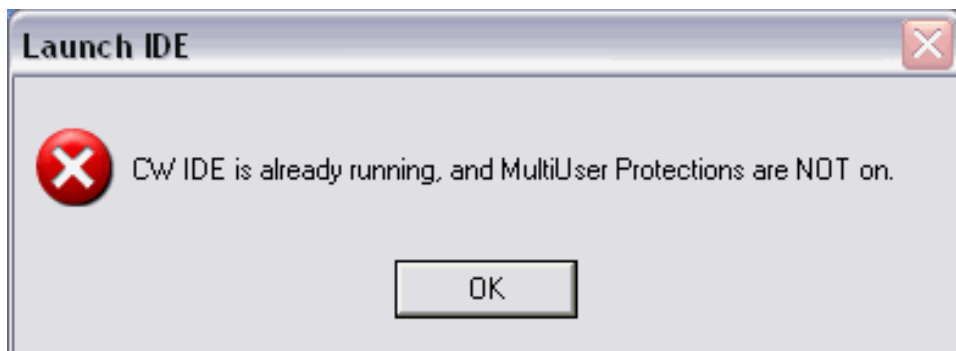
```

After compiling the program, create a shortcut on the desktop for each Clarion version.

- Copy the LaunchIDE.exe to your Windows System Folder
- Right Click on the Desktop and Select New Shortcut.
- In the Target Entry type C:\WINDOWS\LaunchIDE.exe 60EE for Clarion 6 Enterprise Edition.

Now run some tests:

- Make sure that Multi-User Development is not Switched on:
- Run the First Short cut or go to Start Run LaunchIDE 55EE for Clarion 5.5 EE or LaunchIDE 60EE for Clarion 60 EE. If all goes well the first Clarion IDE will run normally
- Trying to run a second copy of the Clarion IDE will display the message in Figure 5.



### Figure 5. The launch error message

Cool, my registry is not trashed...

Since using this utility, I never had crashed my Registry.RTF file. (Note: It's always a good idea to store a backup of your Registry.TRF file somewhere.)

Thanks again to Mark Goldberg for this very handy little utility. Remember, if you make any changes to this program, please [email Mark](#) and send him the updated version.

[Download the source](#)

[Updated source from Mark Goldberg](#)

---

*[Danie de Beer](#) was born and raised in South Africa. He is a qualified BTech Electronic Engineer with a Digital Systems Degree and Higher Educational Diploma in Mathematics and Science (1994). Danie started playing with Spektrum ZX 80 and Commadore 64 computers at an early age. After two years in the Military Service, he started as a Junior Mainframe Programmer in 1988, using Cobol, Natural, Vtam Adabas. He moved to PC programming in 1992 at a cement laboratory, where he converted HP Basic programs to Turbo Basic and Clarion Professional Developer 2.0 (DOS). After being promoted to management and taking a six year hiatus from programming, he moved to the United States where he started programming again using Clarion 5.5 EE. He is now changing over to C6.*

## Reader Comments

[Add a comment](#)

**Randy Rogers pointed out the BeginUnique command in...**  
**Small enhancement suggestions would be command line...**  
**I've posted an updated source zip from Mark...**  
**Am I doing something wrong with this utility? It runs...**

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

## [Clarion Magazine](#)

# Reborn Free

**CLARION**  
*online*

## Clarion News

[Search the news archive](#)

### [Embedded Firebird](#)

Chris Bordeman reports that Firebird is available in an embedded configuration; you don't even have to register a Firebird ODBC driver with Windows. Look for the TestEmbedded folder in the zip. Just distribute the embedded server dll and the odbc dll in the same directory as the exe, and put the CLIENT=C:\myapp\fbembedded.dll in your connect string. Windows sees this and transfers to your DLL, without checking its driver database. Multiple instances of the embedded version can run at once (runs in-process w/ your app and ignores registry settings) and they don't interfere with a full server that may be installed.

*Posted Tuesday, March 30, 2004*

### [xInactivity 1.1](#)

SealSoft's xInactivity v1.1 is now available. There have been some changes in template. Now the setting for run procedure on event looks like a standard procedure definition, i.e. you can just RUN a procedure or START a procedure in a new thread, set procedure parameters, request file action and return a value.

*Posted Tuesday, March 30, 2004*

### [In-Memory Database Driver Information](#)

The Softvelocity Memory driver is a new file driver technology that does not use physical tables for working with data. This is due to a RAM-based technology known as IMDD (In-Memory Database Driver).



All data is stored in Random Access Memory (RAM), which gives the driver a number of unique properties.

*Posted Tuesday, March 30, 2004*

### **[xInactivity Available On ClarionShop.](#)**

xInactivity v1.0 is now available on ClarionShop.

*Posted Tuesday, March 30, 2004*

### **[Advanced Data Dictionary Architect 1.0.2](#)**

Advanced Data Dictionary Architect 1.0.2 is now available. Changes include: fixed parsing error bug in full rebuild mode; Added button which allows changing database connection on the fly from the database synchronizer module; Rollback performed on all the operations done on an entity if the backend returns an error - this adds safety to the upgrading process; Added Windows XP manifests for all the modules. ADDA is a multipurpose toolkit for designing, creating and maintaining the database layer throughout the entire application lifecycle.

*Posted Tuesday, March 30, 2004*

### **[Icetips Previewer 2.0](#)**

Version 2.0 of the Icetips Previewer is now available. The Icetips Previewer is created as a procedure in your application. You can modify it any way you want. It comes with full source code, embedded in the Previewer procedure, ready for you to view and modify. There are no black box DLLs, no external source code files, no class files - just the code in your application. You can even have multiple previewers in the same application and use different previewers for different reports. You can add it to any applications in Clarion 4, 5, 5.5 and 6.0, ABC or Clarion chains. Supports standalone and local compiles, multi DLL projects, etc. Among other new things in version 2.0 are extension templates to allow selecting printers and print directly from the previewer. The main addition is the new Clarion 6 target options so that now you can print to PDF, HTML, XML and Text files directly from the previewer. Version 2.0 also has support for PDF-Tools built in. The manual includes a quick start chapter and a tutorial on how to add the Clarion 6 target options both to your

application and report as well as the Previewer. You can download the documentation alone from <http://www.icetips.com/downloadfile.php?FileID=39> or from the downloads link on the Previewer page. The Icetips Previewer works with many other third party tools, such as Fomin Report builder, CPCS, EasyListPrint, Icetips Xplore, Email-Report from Vivid Help, PDF-Tools and the wPDFControl Wrapper from Klarisoft.

*Posted Tuesday, March 30, 2004*

### **[Multilingual Text To Voice](#)**

Jeff Slarve points out this page which lets you type in text and get synthesized audio. You can also download the audio file. Bob Healy notes that he's used this site to create application messages for errors, welcome, good bye, etc.

*Posted Tuesday, March 30, 2004*

### **[Fenix ASP.NET Generator at ETC 2004](#)**

A Fenix Team (Erik Pepping and Sebastian Talamoni) will be present at ETC 2004, and will present a Fenix ASP.NET Generator demo. They will also be available during the event for one-to-one demonstrations and Q&A. RADventure is also an ETC sponsor, meaning that you can even win a free copy of Fenix if you attend the conference.

*Posted Tuesday, March 30, 2004*

### **[1st Logo Design Spring Specials](#)**

Logo Package Special: Buy any logo package and get free your choice of stationery upgrade, or your new logo in wallpaper and a screen saver. The Developer's Package Special includes: Application Logo Design; Application Icon Design; Splash Screen; Product Box Type 21 Wallpaper. Image Sale - collection includes: 12 high resolution image packages (three to be released on a future date); Free updates for life; 345 icons.

*Posted Tuesday, March 30, 2004*

### **[Clarion Third Party Profile Exchange Updated](#)**

The Clarion Third Party Profile Exchange consists primarily of profiles of third party add-on products and vendors. This includes freeware

templates and tools as well. Online and Downloadable Profiles available. Online product profiles include Product Internet URL, Order URL, Currency code, Dated Price Quote, Grouped by Category, Clarion 6 Compatible, Extended Description and Download Page Reference. Currently, there are 467 product profiles and 461 vendor profiles. You must have Product Scope 32 PRO Version 5.0 to view profiles with data files (downloadable profiles). 276 Clarion 6 compatible products as of this release.

*Posted Tuesday, March 30, 2004*

### **[Firebird Links And Tutorial](#)**

Johan van Zyl has set up a page for Clarion developers interested in using the Firebird (formerly Interbase) open source SQL database. There are several step by step articles, as well as a number of useful links.

*Posted Tuesday, March 30, 2004*

### **[SealSoft xTipOfDay 2.0](#)**

xTipOfDay v2.0 has been released. Changes include: No more black box DLL, just pure Clarion code; Standard TRN file for localization; Supports Window XP manifest; New CHM help file; New installation kits; Some cosmetic changes; Some small bugs were fixed.

*Posted Monday, March 22, 2004*

### **[ClarionForge Input Requested](#)**

ClarionForge is very close to being ready. Ron is looking for input on the Code Snippets, Forum, Trove, and Help Wanted sections.

*Posted Monday, March 22, 2004*

### **[CapeSoft Training At ETC 4](#)**

A CapeSoft Training Session will be held in Gatlinburg on June 8th 2004, from 1 p.m. to 5 p.m. at the ETC 4 venue. Training will be free of charge and will be aimed at those people who already have the following CapeSoft products: NetTalk; Replicate; Insight Graphing. This will not be a sales pitch, but rather a more detailed look at using these tool. Training is open to all ETC 4 attendees and is free of charge. More details will be posted on relevant web sites as they

become available.

*Posted Monday, March 22, 2004*

### **[SB5 Beta Online Forum](#)**

The Lindersoft discussion forum is now using the latest version of vBulletin.

*Posted Monday, March 22, 2004*

### **[Clarion And Armadillo HowTo](#)**

Bernie Groperrin has published a HowTo on using Armadillo and eSellerate with Clarion.

*Posted Monday, March 22, 2004*

### **[DCT2SQL For Interbase](#)**

An updated DCT2SQL template with Interbase improvements is now available for download.

*Posted Monday, March 22, 2004*

### **[MAV Direct ODBC Rewrite](#)**

The MAV library has been completely rewritten order to fix memory problems. All registered users can download the last versions (007 and 004 from March, 09). Also there is a new test procedure in ABCMAVT.APP which emulates in a cycle the MAVLOAD calls of all types. Updated demo and benchmark applications.

*Posted Monday, March 22, 2004*

### **[Gitano Buy One Get One Free](#)**

For a limited time buy any product from Gitano Software and get another one of the same value or less for free.

*Posted Tuesday, March 16, 2004*

### **[xDataBackup Manager Pro 1.7](#)**

New in this release of xDataBackup Manager: Option to include or not include Windows XP manifest file when you turn On checkbox "Program will work under Windows XP"; "Remove icons" option.

Updated Documentation, Demonstration program and Installation Kits for Clarion 5, Clarion 5.5 and Clarion 6 are available.

*Posted Tuesday, March 16, 2004*

### **[Free Web Email Link Cloaking Utility](#)**

Ben Brady has released a free web email link cloaking utility to prevent the harvest of email addresses from web pages.

*Posted Tuesday, March 16, 2004*

### **[Bg XML Dico Video](#)**

Bernie Groperrin has released an experimental video demonstrating Hi guys, as a kind of experiment, I put a video demonstrating simply Bg XML DICO.

*Posted Tuesday, March 16, 2004*

### **[Excel Charts 1.0](#)**

Excel Charts is a simple but powerful tool that helps you to include professional quality charts in your Clarion applications using Microsoft Excel charting features. No previous charting knowledge is required. Features include: More than 90 different chart types; Support for column, bar, line, pie, doughnut, area, scatter, radar, surface, bubble, stock, cylinder, cone and pyramid chart types; 20 predefined chart types included; Allows adding charts directly into any Clarion window; Allows adding charts directly into any Clarion report; Both 2D and 3D charts supported; Single and multiple series charts supported; Allows loading series from queue, view or setting a fixed number of series; Allows loading categories from queue, view or setting a fixed number of categories; Titles, texts and fonts completely configurable; More than 20 different textures that can be applied to your charts; More than 20 predefined gradient styles that can be applied to your charts; Elevation, rotation and perspective of 3-d charts completely configurable; More than 15 different chart areas that can be formatted individually. No black box DLLs. All source included.

*Posted Tuesday, March 16, 2004*

### **[gReg Plus \\$99 Offer](#)**

gReg Plus is on sale for \$99, which is \$200 off the regular price.

*Posted Tuesday, March 02, 2004*

### **[Sticky Notes For C6](#)**

Robert Paresi has made an update of Sticky Notes available for C6. Because of the GAIN:FOCUS regression, the auto-close of the Help/About screen will have to wait until 6.1 is released. This is a free product.

*Posted Tuesday, March 02, 2004*

### **[EasyVersion 2.02](#)**

EasyVersion 2.02 is works only with Clarion 6. Please use version 2.01 with Clarion 5 and 5.5. Changes in this release: Added possibility to add some additional information into CLW modules; Fixed some small bugs.

*Posted Tuesday, March 02, 2004*

### **[File Manager 3 Beta 3.30](#)**

File Manager 3 version 3.30 beta is ready and available for download.

*Posted Tuesday, March 02, 2004*

### **[New Wallpaper Template](#)**

Castle Computer Technologies has released a new wallpaper template which allows your end user to change and/or suppress wallpaper on the fly. Price is \$15.

*Posted Tuesday, March 02, 2004*

### **[Encourager Software Web Site Redesign](#)**

The Encourager Software web site has been redesigned using Xara Webstyle 4 templates, graphic tools, and menus.

*Posted Tuesday, March 02, 2004*

### **[Solace Software ReSort Pro & New Web Site Design](#)**

Simon Burrows has released a new template called ReSort Pro. This is a beefed up version of the free ReSort template. It includes the following enhancements for end users: Sort orders can be saved with a user defined name; On browses users can make sort orders appear as tabs on sheets where browse sorts are controlled by tabbed sheets; On reports sort orders can be appended to or replace the built in sort order; On reports users can define a sort order to be used all

the time for an individual report without the sort order window appearing before each report - this can be overridden by the end user; Uses SQL external names where applicable; Programmer can make certain fields unavailable (i.e. Third Normal Form references to related files); All source code is available; Multi-DLL compatible. ABC only, price around £40 in beta £55 when it goes gold. Also note that the Solace Software web site has been revamped for easier navigation.

*Posted Tuesday, March 02, 2004*

### **[MAV Direct ODBC 0.04 And 0.07](#)**

New releases of MAV Direct ODBC are now available. Changes in library version 0.07 include: Fixed memory leak when working with ANY variables in groups; Added BLOB field support; Improved internal library and classes optimization. Price is \$99 during beta testing, \$249 when it goes gold (free upgrade for beta testers). Clarion 5.0b, 5.5 and 6.0, ABC or Legacy, 32-bits only. Changes in template version 0.04 include: MAVSave class modified for the purpose of checking of the BLOB field modifications on saving a form; improved internal library, classes and templates optimization. Price is \$99 during beta testing, \$149 when it goes gold (free upgrade for beta testers). Requires MAV Direct ODBC library version. Clarion 5.0b, 5.5 and 6.0, ABC or Legacy, 32-bits only.

*Posted Tuesday, March 02, 2004*

### **[Freeware Templates Available](#)**

A number of freeware templates, including OSShield and DriveInfo, are now available for download, courtesy of Roel Abspoel.

*Posted Tuesday, March 02, 2004*

### **[Mail & Merge Manager v1.1](#)**

Mail & Manager v1.1 is now available for download. This version has C6 support and minor improvements.

*Posted Tuesday, March 02, 2004*

### **[Outlook Products Available At ClarionShop](#)**

Softmasters' Outlook-related products are now available through

ClarionShop.

*Posted Tuesday, March 02, 2004*

### **[New Clarion Dictionary Utility](#)**

BG Softfactory Inc. has released BG XML Dico, a utility that lets you browse your Clarion dictionary in a new way. This product uses Clarion templates, COM, XSLT, and HTML with Javascript. A utility template generates an XML file from the current application dictionary. Then, this file is loaded as a DOM (Document Object Model) into the Microsoft XML parser, where it is then transformed four times by four different XSL style sheets; the result is finally displayed via a Web browser control. No TXDs are generated and/or used. The first transformation creates the html frame in which each one of the three other transformations will be displayed. The second transformation generates the header, and calculates the statistics. The third transformation generates the JavaScript tree menu. The fourth generates the dictionary data, organizing file attributes, keys, fields and relations in clear, easy to read tables, with hyper links between related files. The introductory price of \$39 is good until March 31, 2004, after which the price goes up to \$50.

*Posted Tuesday, March 02, 2004*

### **[O'Reilly Launches New Windows DevCenter](#)**

O'Reilly's new Windows DevCenter features in-depth articles geared to developers and system administrators, plus plenty of tips, tricks, and reviews for Windows power users. Current articles include "Kicking the Tires of XP Service Pack 2," "Tuning Automatic Updates," and "Getting Started with Microsoft InfoPath 2003." Ongoing coverage will address .NET, VBScript, SQL, system administration, wireless, and emerging Windows issues. The daily news section offers the latest stories in the Windows world, and weblogs by notable Windows luminaries and O'Reilly authors present a range of expert opinions on Windows issues.

*Posted Tuesday, March 02, 2004*



Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

## [Clarion Magazine](#)

online sales and delivery  
for your applications & tools

Developer **PLUS**

# Validating Credit Card Numbers

by **Abe Jimenez**

Published 2004-03-02

I recently needed to incorporate credit card validation into a Clarion application. I looked in Clarion Magazine and was very surprised when I could not find any articles on the topic. It seems to me that this is a common task for anyone who develops a sales application, and I was hopping to find a ready-made function. Since I had to write my own, I decided to share it with the rest of the Clarion Magazine subscribers.

## The Objectives

The function I describe in this article meets the following objectives:

- I can easily call the function from anywhere I need to validate a credit card
- The function will return the type of credit card
- If the card number is invalid, the function will advise me that a bad card number has been entered.

The function will be called as follows:

```
ReturnCode = ValidateCreditCardNumber(CreditCardNumber)
```

After calling the above function, I want to be able to use the return code to either display an error message or to display the type of credit card entered.

## The Algorithm

Each credit card company has prefixes they use in their credit card numbers, as follows:

<b>Card</b>	<b>Prefix</b>	<b>Digits</b>
Master Card	Starts with 51,52,53,54 or 55	16
Visa	Starts with 4	13 or 16
American Express	Starts with 34 or 37	15
Diners Club/ Carte Blanche	Starts with 300,301,302,303,304,305,36 or 38	14
Discover	Starts with 6011	16
JCB	Starts with 3 (but not followed by the digits which are valid for American Express or Diners Club), 1800, or 2131	15 or 16

In order for a card number to be valid it must pass a MOD 10 validation, as follows:

1. The credit card number is reversed.
2. Each of the digits is processed into the mod 10 formula. The odd digits (1<sup>st</sup>, 3<sup>rd</sup>, 5<sup>th</sup>, etc.) are added into a total. The even number digits (2<sup>nd</sup>, 4<sup>th</sup>, 6<sup>th</sup>, etc) are first multiplied by 2 and then each of the digits in the resulting number are added into the same mod 10 total.
3. If the mod 10 total is a multiple of 10, then the credit card is valid.

This is kind of complicated to understand, so I have included the table below to illustrate the process. The example validates credit card number 4123456789012.

<b>Digit</b>	<b>1<sup>st</sup></b>	<b>2<sup>nd</sup></b>	<b>3<sup>rd</sup></b>	<b>4<sup>th</sup></b>	<b>5<sup>th</sup></b>	<b>6<sup>th</sup></b>	<b>7<sup>th</sup></b>	<b>8<sup>th</sup></b>	<b>9<sup>th</sup></b>	<b>10<sup>th</sup></b>	<b>11<sup>th</sup></b>	<b>12<sup>th</sup></b>	<b>13<sup>th</sup></b>	<b>Result</b>
<b>Original</b>	4	1	2	3	4	5	6	7	8	9	0	1	2	
<b>Reverse</b>	2	1	0	9	8	7	6	5	4	3	2	1	4	

<b>Calculation</b>	2	2*1	2	2*9	8	2*7	6	2*5	4	2*3	2	2*1	4	
<b>Mod 10</b>	2	+2	+2	+1+8	+8	+1+4	+6	+1+0	+4	+6	+2	+2	+4	53
<b>Total</b>														

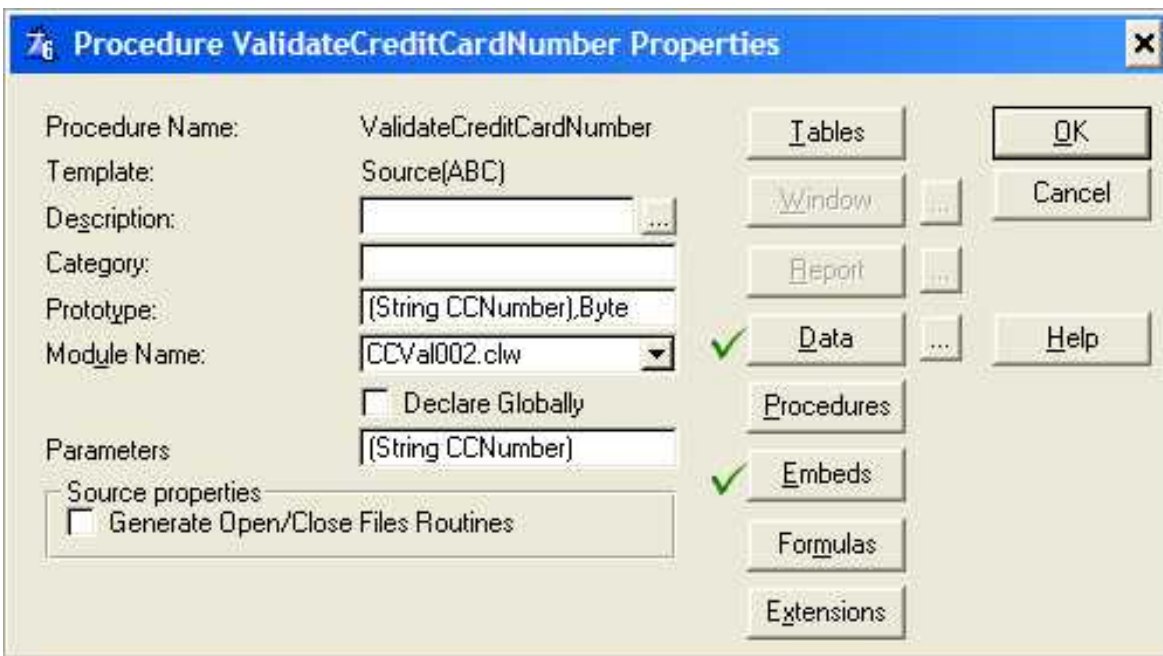
The row labeled **Original** contains each digit in the credit card number. The row labeled **Reverse** shows the reversed string. The **Calculation** row shows that the odd numbered digits are left alone and the even numbers are multiplied by 2.

The bottom row shows the numbers that are added into the total. Notice the column labeled 4<sup>th</sup>. The **Calculation** row shows  $2*9$ , which is 18. Each digit is added into the total separately ( $1 + 8$ ). Columns 6 and 8 are processed similarly. The result column shows the total value you get when you sum up columns 1<sup>st</sup> through 13<sup>th</sup>.

If this sum (53) were a multiple of 10, the card number would be valid. In this example, the card number is invalid. If the total had been 50 or 60, the number would have been a valid Visa card number, since it starts with a 4.

## The Clarion Translation

Now I need to translate the above logic into a Clarion function. I'm going to create a new source procedure. The first thing is to prototype the function as shown in Figure 1.



**Figure 1. Creating the function as a source procedure**

Then I need to define some variables:

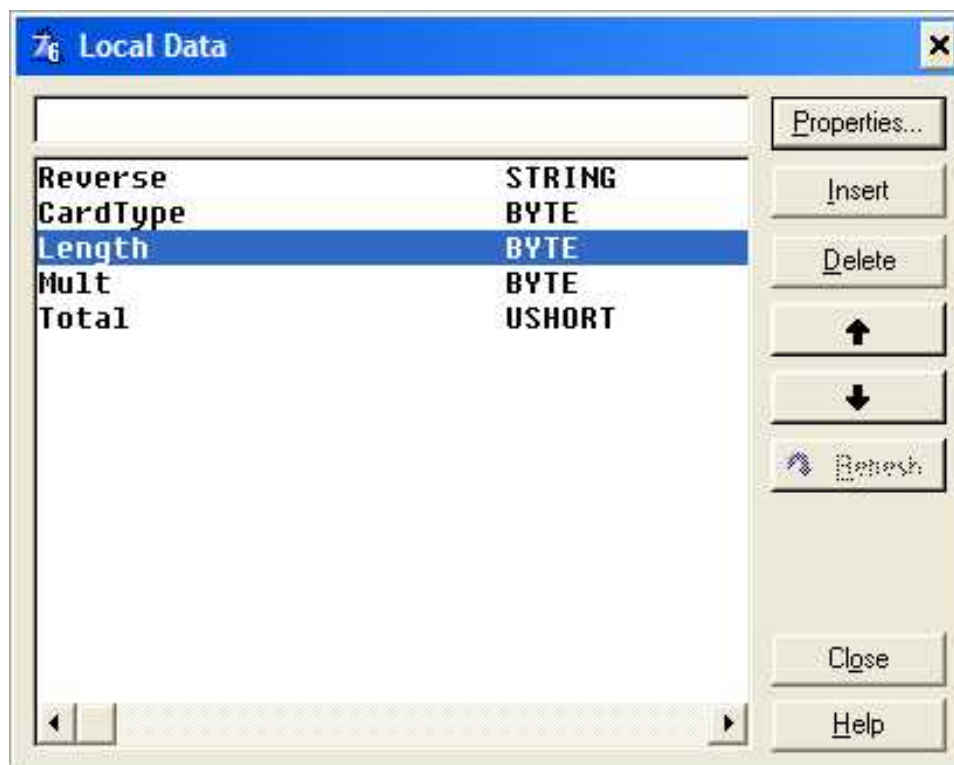
`Reverse` is a string the same length as the credit card number. It will be used to store the credit card digits in reverse order.

`CardType` is the return code of the function. A return code of 0 will indicate that the user has entered an invalid credit card number. Return codes 1-6 will mean Master Card, Visa, American Express, Diners Club, Discover and JCB respectively.

`Length` is a byte I will use to store the length of the credit card string during validation.

`Mult` is a byte I will use to temporarily store the result of multiplying the even digits in the credit card number by 2.

`Total` is a `USHORT` I will use to store the sum of the digits. You can define these variables in the Local Data embed or by pressing the data button in the procedure as I chose to do. Figure 2 is a screen print of the variable definitions.



**Figure 2. Adding the procedure data**

All the code will be placed in the Processed Code embed of the procedure.

There are five steps to the code:

1. Get the card type:
2. Check that all the characters are numeric
3. Validate the length of the card based on its type
4. Perform the Mod 10 validation on the number.
5. Return the result of my validation to the calling procedure.

This translates into Clarion routines as follows:

```

Do GetCardType
If CardType then Do CheckNumeric.
If CardType then Do ValidateLength.
If CardType then Do Mod10Validation.
Return(CardType)

```

If after executing any one of the first four lines above, the `CardType` variable is 0, it means that the validation failed, in which case I don't do any more checking and I simply return 0 (meaning Invalid Card). If on the other hand a test passes, the code performs the next test.

The `GetCardType` routine is as follows. Each `IF` statement checks the starting

characters of the card number and sets the `CardType` variable accordingly. If a card number does not start with any of the valid values, the `CardType` variable is set to 0, which means that an invalid card number was entered.

GetCardType Routine

```

If      CCNumber[1:2] = '51'      !Master Card
      CardType = 1
ElsIf  CCNumber[1:2] = '52'      !Master Card
      CardType = 1
ElsIf  CCNumber[1:2] = '53'      !Master Card
      CardType = 1
ElsIf  CCNumber[1:2] = '54'      !Master Card
      CardType = 1
ElsIf  CCNumber[1:2] = '55'      !Master Card
      CardType = 1
ElsIf  CCNumber[1:1] = '4'       !Visa
      CardType = 2
ElsIf  CCNumber[1:2] = '34'      !Amex
      CardType = 3
ElsIf  CCNumber[1:2] = '37'      !Amex
      CardType = 3
ElsIf  CCNumber[1:3] = '300'     !Diners Club
      CardType = 4
ElsIf  CCNumber[1:3] = '301'     !Diners Club
      CardType = 4
ElsIf  CCNumber[1:3] = '302'     !Diners Club
      CardType = 4
ElsIf  CCNumber[1:3] = '303'     !Diners Club
      CardType = 4
ElsIf  CCNumber[1:3] = '304'     !Diners Club
      CardType = 4
ElsIf  CCNumber[1:3] = '305'     !Diners Club
      CardType = 4
ElsIf  CCNumber[1:2] = '36'      !Diners Club
      CardType = 4
ElsIf  CCNumber[1:2] = '38'      !Diners Club
      CardType = 4
ElsIf  CCNumber[1:4] = '6011'    !Discover
      CardType = 5
ElsIf  CCNumber[1:1] = '3'       !JCB
      CardType = 6
ElsIf  CCNumber[1:4] = '1800'    !JCB
      CardType = 6
ElsIf  CCNumber[1:4] = '2131'    !JCB
      CardType = 6
Else
      CardType = 0                !Bad Card
End

```

**The CheckNumeric routine is very simple. It simply sets the `CardType` to 0 if the `CCNumber` string contains any characters that are not allowed. You will find that many of your users will enter credit card numbers with dashes or spaces**

separating the digits. Most credit card processing software will expect numeric strings, and dashes or spaces will cause an error. The application my client uses absolutely does not like anything but numbers.

In some cases you may just want the program to perform the card validation ignoring non-numeric characters instead of returning them as invalid. If that's the case, you can simply not code this routine. I will show you later how to accomplish this.

CheckNumeric Routine

```
! If you want the program to simply ignore the non-numeric
! characters, leave this routine out and add in the two optional lines of code
! as shown in the ReverseTheNumber routine
If Not Numeric(Clip(CCNumber))
    CardType = 0
End
```

The ValidateLength routine is also very simple. It assigns the length of the string to the Length variable. The CardType variable starts off with a value from 1 to 6 indicating which credit card type I'm checking (from the GetCardType routine previously called). The execute statement will validate the length to be the appropriate size for the card. If it is not, the CardType variable is set to 0.

ValidateLength Routine

```
Length = Len(Clip(CCNumber))
Execute CardType
    If not Length = 16 then CardType = 0. ! 1=Visa
    If not (Length = 13 Or Length = 16) then CardType = 0. ! 2=MC
    If not Length = 15 then CardType = 0. ! 3=Amex
    If not Length = 14 then CardType = 0. ! 4=Diners
    If not Length = 16 then CardType = 0. ! 5=Discover
    If not (Length = 15 Or Length = 16) then CardType = 0. ! 6=JCB
End
```

The last step is to do the Mod 10 validation. Here's the routine, which simply calls a couple of other routines and does one final test:

Mod10Validation Routine

```
Do ReverseTheNumber
Do TotalUp
If Not (Total % 10 = 0) then CardType = 0.
```

The first step is to reverse the string:



## ReverseTheNumber Routine

```
Reverse = ''
Loop i# = Len(Clip(CNumber)) to 1 by -1
  If numeric(CNumber[I#]) ! only if not using CheckNumeric routine
    Reverse = Clip(Reverse) & CNumber[I#]
  End
  ! only if not using CheckNumeric routine
End
```

**Note:** The third and fifth line of the routine are only necessary if you are not using the `CheckNumeric` routine, which determines that a card number is invalid if it contains anything but numbers.

The code ensures that the variable is cleared, and then loops through the original string backwards appending the characters to the reversed string. I only wanted to use Clarion code in this article, but it would have been more efficient and just as easy to use the `StrRev` standard C function in Clarion's run time library. The prototype for `StrRev` is:

```
StrRev(*cstring),cstring,raw,name('_strrev')
```

The above line is straight out of the Clarion help text. If you are interested, just do a search for `StrRev`.

Here's the code for the `TotalUp` routine:

## TotalUp Routine

```
Total = 0
Loop i# = 1 to Len(Clip(Reverse)) by 1
  ! 1st, 3rd, 5th ... digit in reverse card number
  If Not i# % 2 = 0
    Mult = Reverse[I#]
  Else
    ! 2nd, 4th, 6th ... digit in reverse card number
    Mult = Reverse[I#] * 2
  End
  If Mult < 10
    ! Single digit
    Total += Mult
  Else
    ! Two digit
    Total += (Int(Mult/10) + (Mult % 10))
  End
End
```

This routine loops through each character of the reversed string. In the first `IF` statement the `Mult` variable is assigned the value of the digit for odd numbered positions in the string. For even numbered digits, `Mult` is assigned the character value times two. At the end of this step, `Mult` can have a value of

0 through 18.

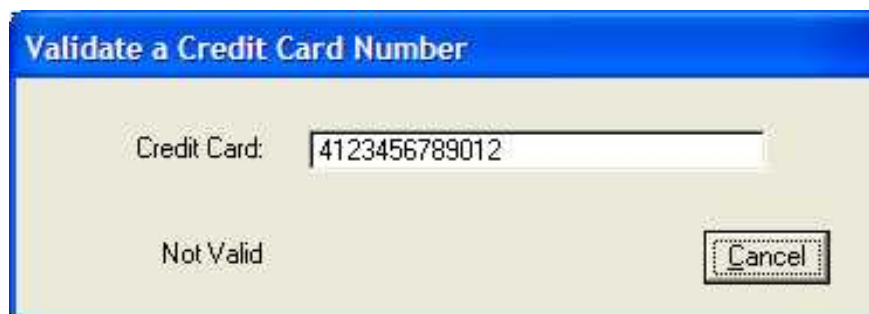
The second `IF` statement accumulates the value of `Mult` into the total. If the value is less than 10 (0-9), a single digit is added to the total. If the value is two digits long, each digit is added separately. For example, 14 would result in adding 1 and 4 to the total.

The last line in the `Mod10Validation` routine changes the value of `CardType` to 0 (invalid) if the total is not a multiple of 10.

Now that all the validations have been done, the function issues `Return(CardType)` to send the result back to the calling procedure:

## Using the function

To demonstrate how to use the function, I created a simple window as shown in Figure 3.



**Figure 3. Testing the code**

The `ValidateCreditCardNumber` function is called in the accepted embed of the credit card number field. If you are using Clarion 6, you can create a validation rule to execute the code. I wanted to make the example work with older versions of Clarion, so I didn't do this. Here's the embed code:

```
ReturnCode = ValidateCreditCardNumber(CreditCard)
Execute ReturnCode + 1
    CardType = 'Not Valid'
    CardType = 'Master Card'
    CardType = 'Visa'
    CardType = 'American Express'
    CardType = 'Diners Club'
    CardType = 'Discover'
    CardType = 'JCB'
End
```

**Note:** The scope of the code in this article is to provide a mechanism for validating a credit card number and determining the type of credit card. There is a lot more to validate in a credit card transaction. For example, you should make sure that the card is not expired and that the cardholder's name is entered by the user. Some credit card terminal programs require that transactions be coded as Card Not Present, Card Present, or Card Not Present Internet. You can also enter the street address and zip code for address validation, and you usually need to provide some kind of code to identify the type of transaction (Authorize-Only or Funds-Capture). Also your clients may only accept some of the credit card types and not accept others. Most of my clients accept Visa, Master Card, Discover and American Express, and do not accept Diners Club/Carte Blanche and JCB. If enough readers are interested, I will write a follow-up article on the additional validation required.

I hope you find this code useful. I have included a sample app with the code printed on this article. In addition to the credit card validation algorithm and a function you can use in your own applications, the sample code shows some examples of string slicing techniques.

[Download the source](#)

---

*[Abe Jimenez](#) started programming in the late 70s in RPG on the IBM System 34. Towards the late 80s he began using Clarion 2.1 for DOS. Over the years he has programmed in all versions of Clarion, but not continuously. He is now an independent consultant and programs PC applications in Clarion and AS400 applications in RPG.*

## Reader Comments

[Add a comment](#)

**Nice article and source, should be very helpful to many of...**

**For a C# Web Service version.....**

**Very nice article. If anyone is interested the algorithm...**

**Please include ASCII text file source. The app is V6, and...**

**To download a C55 version of the app go to...**

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

## [Clarion Magazine](#)



### **A Class Wrapper for Brice Schagane's Menu Buttons**

**by Nik Johnson**

Published 2004-03-26

My copy of the new [Clarion Tips & Techniques book](#) arrived as I was struggling with an entry screen problem. I immediately seized on the opportunity do something more pleasant, all the while telling myself that reading technical books is also work.

Flipping through the pages, it was soon apparent that there was a lot of meat here. While I could certainly sample articles at random, I was sure that I'd miss something if I didn't approach the book in a systematic way. The system I chose was simple: Start at the beginning.

The first article, by Brice Schagane, presented a method for [adding a menu to a button](#). What a coincidence! My entry screen problem was that I was clean out of real estate, and here, on page 3, was the answer. Damn! That Dave Harms guy is amazing. How did he know?

Figure 1 shows Brice's menu button in action.



**Figure 1. Brice Schagane's menu button**

After reading the article and playing with the code a bit, I realized that every time I used this technique I'd have to come back to the article and learn how to do it all over again. There had to be an easier way, and, in Clarion, an easier way is usually spelled t-e-m-p-l-a-t-e or c-l-a-s-s. I chose the latter.

## A Simple Wrapper

One nice thing about classes is that you can do a lot of prototyping without writing a lot of code. I started simple:

```

MenuButtonQueue QUEUE,TYPE
Caption          CSTRING(41) ! For popup menu
                END

RECT            GROUP,TYPE
Left           SIGNED
Top           SIGNED
Right          SIGNED
Bottom         SIGNED
                END

HANDLE         EQUATE(UNSIGNED)
HWND           EQUATE(HANDLE)
CAIMenuButton CLASS,TYPE,MODULE('CAIABC.CLW'), |
                LINK('CAIABC.CLW',_ABCLinkMode_), DLL(_ABCDllMode_)
MBQ            &MenuButtonQueue
ButtonHandle   HWND
ButtonControl  LONG
Rectangle      LIKE(RECT)
Construct      PROCEDURE
Destruct       PROCEDURE

```

```
Init          PROCEDURE(LONG pButton)
AddItem       PROCEDURE(STRING pCaption)
Popup        PROCEDURE,BYTE
            END
```

This takes care of all of Brice's code except the API prototype, which I included in the map at the beginning of CAIABC.CLW.

Using the class is easy:

1. Add a button to the window, using Brice's instructions. I'll call it ?Bth:Actions as Brice did.
2. In the procedure data section, instantiate a menu button object:

```
MB CAIMenuButon
```

3. In ThisWindow.Init, after the window is opened, initialize the object:

```
MB.Init(?Bth:Actions)
MB.AddItem('&Create a New Invoice')
MB.AddItem('&View/Edit Selected Invoice')
MB.AddItem('&Delete Selected Invoice')
```

4. In the ThisWindow.TakeAccepted embed for ?Bth:Actions, enter the following:

```
EXECUTE MB.Popup()
    POST(EVENT:Accepted,?Insert)
    POST(EVENT:Accepted,?Change)
    POST(EVENT:Accepted,?Delete)
END
```

That's it: Simple, straightforward, and easy enough not to require a template.

One problem: It doesn't work in my situation.

## A Variable-Length Menu

In the case at hand I need to make particular actions available depending on the characteristics of the record being processed. One way to do this would be to make the AddItem calls conditional:

```
IF Condition1
    MB.AddItem('&Create a New Invoice')
END
IF Condition2
    MB.AddItem('&View/Edit Selected Invoice')
END
IF Condition3
    MB.AddItem('&Delete Selected Invoice')
END
```

The obvious problem with this arrangement is that the integer returned by `MB.Popup()` is no longer useful.

Allowing `MB.AddItem` to specify the integer to be returned solves this problem. This requires an expansion of `MenuButtonQueue` and a change in the prototype of the `AddItem` method:

```
MenuButtonQueue QUEUE,TYPE
Caption          CSTRING(41) ! For popup menu
Sequence        BYTE         ! For EXECUTE statement
                END

AddItem          PROCEDURE(BYTE pSequence,STRING pCaption)
```

With these changes in place, the code at `ThisWindow.TakeAccepted` becomes:

```
IF Condition1
  MB.AddItem(1,'&Create a New Invoice')
END
IF Condition2
  MB.AddItem(2,'&View/Edit Selected Invoice')
END
IF Condition3
  MB.AddItem(3,'&Delete Selected Invoice')
END
```

## Getting Fancy

While poking through the LRM, double checking the operation of `POPUP()`, I noticed that menu selections, as well as buttons, can have icons. Since icons can provide a nice touch in many situations, I decided to add this capability to the class. The changes were minimal:

```
MenuButtonQueue QUEUE,TYPE
Caption          CSTRING(41) ! For popup menu
SmallIcon       CSTRING(41) ! For popup menu
Sequence        BYTE         ! For EXECUTE statement
                END

AddItem          PROCEDURE(BYTE pSequence,STRING pCaption,<STRING pSmallIcon>)
```

I chose the name "SmallIcon" to remind me that the space available on a menu item doesn't lend itself to the same kind of icon one would put on a button.

## Boundary Conditions

Programming problems seem to cluster at the edges. That is, it's not the processing in



the middle of the loop but the handling of the first and last iterations that trips us up. In this case there are two boundary conditions to consider:

1. When there is only one active menu selection, pressing the button presents that selection. While this gets the job done, it requires an unnecessary extra mouse click. A better option might be to make the button itself represent the single menu item.
2. When there are no active menu selections, the menu button should probably disappear.

Handling the first of these conditions requires the addition of a few omittable parameters, and provision in `MemoryButtonQueue` to store them:

```

MenuButtonQueue QUEUE,TYPE
Caption          CSTRING(41)  ! For popup menu
SmallIcon       CSTRING(41)  ! For popup menu
Sequence        BYTE          ! For EXECUTE statement
ShortCaption    CSTRING(41)  ! For button (if only one selection)
Icon           CSTRING(41)  ! For button (if only one selection)
                END

AddItem         PROCEDURE(BYTE pSequence,STRING pCaption,<STRING pSmallIcon>, |
                <STRING pShortCaption>,<STRING pIcon>)

```

While buttons can handle larger icons, they tend to be limited in text, hence the designation `ShortCaption` for the text to appear on a button. Both boundary conditions require that the class know when the last `AddItem` call has been made. I chose to handle this with a `SetVisibility` method:

```
SetVisibility PROCEDURE,VIRTUAL
```

`SetVisibility` is called after the last `AddItem`, at which time the standard logic will hide the button if no menu options are present, set the button to match the only option present, or set up the menu if multiple options are present. I made it `VIRTUAL` so that I can override the standard logic when necessary.

## Dynamic Menu Definition

At this point I had but one unsolved problem. It is possible that the conditions controlling the menu selections will change while the screen is open. In this case I will need to repeat the initialization sequence.

Creation and disposal of the queue is handled by `Construct` and `Destruct`, so I didn't need a `Kill` method. If the `Init` method merely `FREES` the queue before any `AddItem` method calls occur, I can reinitialize the whole menu setup at will — almost.

The "almost" part derives from the fact that after any setup which changes the caption

and icon on the button, I've lost the original caption and icon for that button. So I need to provide that explicitly. I chose to modify the `Init` calling sequence for this purpose:

```
Init PROCEDURE(LONG pButton,<STRING pDefaultCaption>,<STRING pDefaultIcon>)
```

## Putting it All Together

Up to this point I had been thinking through the characteristics of the class by using it. To be able to compile and test, I wrote skeleton code for every method. I also had rudimentary code in some methods, allowing me to test the basic setup. But for the most part, the methods were grossly incomplete.

Before investing much in method code, I had arrived at a class definition in which I had a great deal of confidence:

```
MenuButtonQueue QUEUE,TYPE
Caption          CSTRING(41)  ! For popup menu
SmallIcon       CSTRING(41)  ! For popup menu
Sequence        BYTE
ShortCaption    CSTRING(41)  ! For button (if only one selection)
Icon            CSTRING(41)  ! For button (if only one selection)
                END
RECT            GROUP,TYPE
Left            SIGNED
Top            SIGNED
Right          SIGNED
Bottom         SIGNED
                END
HANDLE         EQUATE(UNSIGNED)
HWND           EQUATE(HANDLE)

CAIMenuButton  CLASS,TYPE,MODULE('CAIABC.CLW'), |
                LINK('CAIABC.CLW',_ABCLinkMode_),DLL(_ABCDllMode_)
MBQ            &MenuButtonQueue
ButtonHandle   HWND
ButtonControl  LONG
Rectangle      LIKE(RECT)
DefaultIcon    CSTRING(41)    ! For button (if more than one selection)
DefaultCaption CSTRING(41)    ! For button (if more than one selection)
NoPopup        BYTE          ! If only one selection
Construct      PROCEDURE
Destruct       PROCEDURE
Init           PROCEDURE(LONG pButton,<STRING pDefaultCaption>,<STRING pDefaultIcon>)
AddItem        PROCEDURE(BYTE pSequence,STRING pCaption,<STRING pSmallIcon> , |
                <STRING pShortCaption>,<STRING pIcon>)
SetVisibility  PROCEDURE,VIRTUAL
Popup          PROCEDURE,BYTE
                END
```

Given a good class definition, the methods were easy:

```

MAP
  MODULE( 'Windows.DLL' )
    GetWindowRect( HWND, *RECT), PASCAL, RAW
  END
END

CAIMenuButton.Construct PROCEDURE
  CODE
  SELF.MBQ &= NEW MenuButtonQueue

CAIMenuButton.Destruct PROCEDURE
  CODE
  DISPOSE( SELF.MBQ)

CAIMenuButton.Init PROCEDURE( LONG pButton, <STRING pDefaultCaption>, |
                                <STRING pDefaultIcon> )
  CODE
  SELF.ButtonHandle = pButton{ Prop: Handle }
  SELF.ButtonControl = pButton
  CLEAR( SELF.Rectangle )
  IF OMITTED( pDefaultCaption )
    SELF.DefaultCaption = SELF.ButtonControl{ Prop: Text }
  ELSE
    SELF.DefaultCaption = CLIP( pDefaultCaption )
  END
  IF OMITTED( pDefaultIcon )
    SELF.DefaultIcon = SELF.ButtonControl{ Prop: Icon }
  ELSE
    SELF.DefaultIcon = CLIP( pDefaultIcon )
  END
  FREE( SELF.MBQ )
  SELF.NoPopup = False

CAIMenuButton.AddItem PROCEDURE( BYTE pSequence, STRING pCaption, |
                                  <STRING pSmallIcon>, <STRING pShortCaption>, <STRING pIcon> )
  CODE
  SELF.MBQ.Sequence = pSequence
  GET( SELF.MBQ, SELF.MBQ.Sequence )
  IF NOT ERRORCODE( )
    ASSERT( False, 'Menu entry sequence not unique' )
  END
  CLEAR( SELF.MBQ )
  SELF.MBQ.Caption = CLIP( pCaption )
  IF NOT OMITTED( pIcon )
    SELF.MBQ.SmallIcon = CLIP( pSmallIcon )
  END
  IF NOT OMITTED( pShortCaption )
    SELF.MBQ.ShortCaption = CLIP( pShortCaption )
  END
  IF NOT OMITTED( pIcon )
    SELF.MBQ.Icon = CLIP( pIcon )
  END
  SELF.MBQ.Sequence = pSequence
  ADD( SELF.MBQ, SELF.MBQ.Sequence )
CAIMenuButton.SetVisibility PROCEDURE
  CODE
  EXECUTE RECORDS( SELF.MBQ ) + 1
  SELF.ButtonControl{ Prop: Hide } = True ! No selections: hide button
  BEGIN                                ! One selection: no menu

```

```

    GET(SELF.MBQ,1)
    ASSERT(NOT ERRORCODE(),'Popup choice does not exist')
    IF SELF.MBQ.ShortCaption
        SELF.ButtonControl{Prop:Text} = SELF.MBQ.ShortCaption
    ELSE
        SELF.ButtonControl{Prop:Text} = SELF.MBQ.Caption
    END
    IF SELF.MBQ.Icon
        SELF.ButtonControl{Prop:Icon} = SELF.MBQ.Icon
    ELSE
        SELF.ButtonControl{Prop:Icon} = ''
    END
    SELF.NoPopup = True
    SELF.ButtonControl{Prop:Hide} = False
END
ELSE
    BEGIN                ! More than one selection: menu
        SELF.ButtonControl{Prop:Text} = SELF.DefaultCaption
        SELF.ButtonControl{Prop:Icon} = SELF.DefaultIcon
        SELF.ButtonControl{Prop:Hide} = False
        SELF.NoPopup = False
        DISPLAY(SELF.ButtonControl)
    END
END
CAIMenuButton.Popup PROCEDURE
ControlID LONG
MenuString CSTRING(1001)
MenuChoice BYTE
xpos      LONG
ypos      LONG
width     LONG
height    LONG
ix        BYTE
CODE
IF SELF.NoPopup
    GET(SELF.MBQ,1)
    ASSERT(NOT ERRORCODE(),'Popup choice does not exist')
ELSE
    ControlID = CREATE(0,Create:Group)
    ControlID{Prop:Text} = ''
    ControlID{Prop:Boxed} = True
    ControlID{Prop:Bevel,1} = -1
    ControlID{Prop:Scroll} = True
    GETPOSITION(SELF.ButtonControl,xpos,ypos,width,height)
    SETPOSITION(ControlID,xpos,ypos,width,height)
    GetWindowRect(SELF.ButtonHandle,SELF.Rectangle)
    MenuString = ''
    LOOP ix = 1 TO RECORDS(SELF.MBQ)
        GET(SELF.MBQ,ix)
        IF ix > 1; MenuString = MenuString & '|';END
        IF SELF.MBQ.Icon
            MenuString = MenuString & '[' & PROP:Icon & '(' & |
                SELF.MBQ.SmallIcon&')]'&SELF.MBQ.Caption
        ELSE
            MenuString = MenuString & SELF.MBQ.Caption
        END
    END
    UNHIDE(ControlID)
    MenuChoice = POPUP(MenuString,SELF.Rectangle.Left,SELF.Rectangle.Bottom)

```

```
DESTROY(ControlID)
IF MenuChoice
  GET(SELF.MBQ,MenuChoice)
  ASSERT(NOT ERRORCODE(),'Popup choice does not exist')
ELSE
  SELF.MBQ.Sequence = 0
END
END
RETURN(SELF.MBQ.Sequence)
```

## Conclusion

My screen with its new menu buttons is now in production. User reaction has been very positive. Better yet, I no longer have to worry about the next addition to that screen. The real estate issue is gone, and I have full control over the options presented.

Thanks to Brice Schagane for researching the Windows API and coming up with this very powerful technique. Thanks also to Dave Harms for putting Brice's article on Page 3 where I couldn't possibly miss it.

[Download the source](#)

---

*[Nik Johnson](#) stumbled into the programming racket in 1959 when his boss at Grumman Aircraft insisted on his attending a Fortran class. Since 1986 he has been using Clarion to help clients solve information handling problems.*

## Reader Comments

[Add a comment](#)

**[Why not use the ABC PopupClass? What is the difference?  
I believe you could call Popup.Ask\(\) from a button and get...  
Could you publish the .app, .prj or .clw files as 5.5?](#)**

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

## [Clarion Magazine](#)



### **Using Client-Side Triggers In Clarion 6**

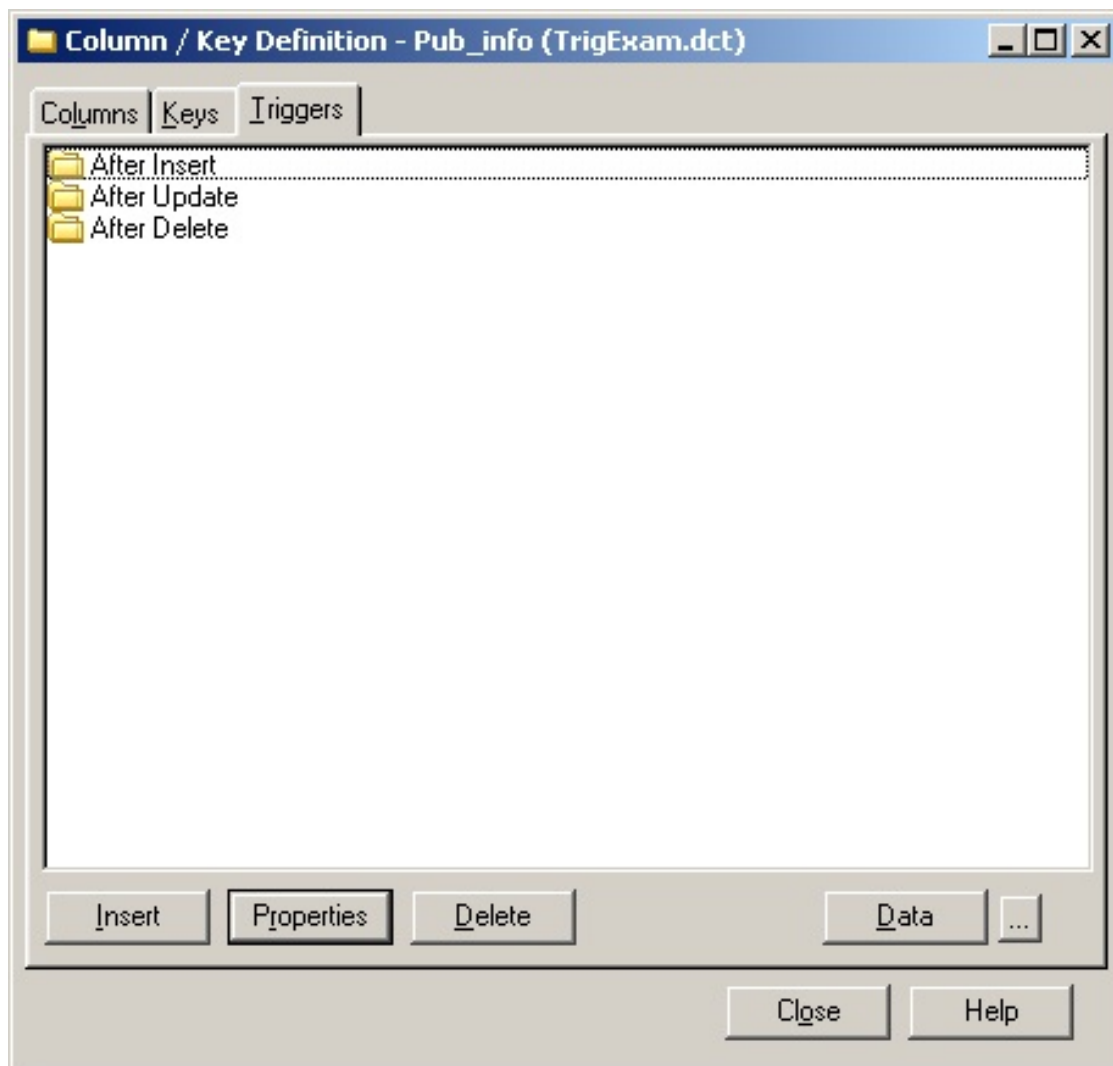
**by Tom Giles**

Published 2004-03-26

Client-side triggers are a new, very neat and useful feature that has been added to Clarion 6, both Professional and Enterprise versions. In addition, it is very easy to use. I guess you wouldn't expect any less since it is in Clarion. Client-side triggers are rules, added to the dictionary, expressed as Clarion code that is executed when a file (table) is accessed. This code can run before and/or after the Add. You enter the code, say for error checking, field validation, computed values etc., and then specify when to invoke the code. At the specified time, your code will be automatically called.

There is an example included with Clarion 6. Documentation is sparse, really non-existent, but once you know the secret, client-side triggers are very easy to use. To use the demo, start Clarion 6, then click on the Pick icon, select the Dictionary tab, click on the Open button and navigate to the Examples folder. In the Triggers folder select and Open the TrigExam.Dct.

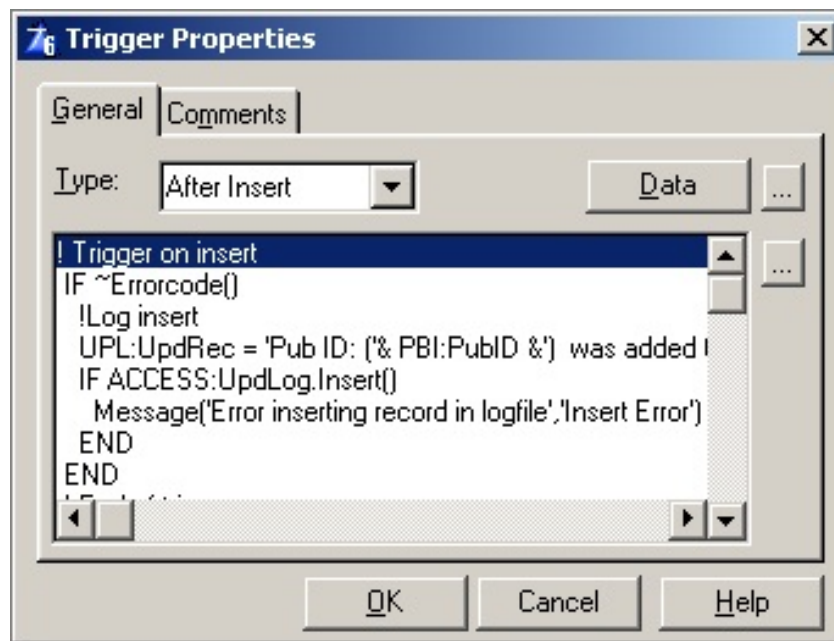
Highlight and double click on the Pub\_info file in the Tables pane to bring up the records. Note there is a new Triggers tab on the Column/Key Definition screen.



**Figure 1. The Triggers tab**

Click on the Triggers tab, then on one of the three folders shown. A trigger Properties form appears, as in Figure 2.





**Figure 2. Trigger properties**

On the General tab specify the timing using the Type: drop down list. In the box below enter your code.

Click on the [...] button at the right to bring up the edit screen and enter any code desired, just as you would in any code embed point. There is also a Data button that will let you enter a new data field. The [...] button has the same function as in the procedure Data button – it exposes the actual declaration.

The Data section is for variables for your trigger code. These are separate from your normal Data fields in the Properties worksheet, which means you can use the same names. This is probably not a good idea unless you use a prefix, since it may be confusing to you later, but the compiler won't complain. I use a TRI:(gger) prefix for these to distinguish these variables from my LOC:(al), MOD:(ule) or GLO:(bal) variables. The TRI variables will not show up in the normal Module/Source files but instead in the ???bc0.clw files. This is because the triggers are generated as part of the file maintenance code, not the procedure code.

Another way to access Triggers is from the main Dictionary screen. Highlight the desired file then click on menu item Edit, then Triggers or highlight the file and press Ctrl+Alt+T. Either of these will take you

directly to the Triggers entry screen.

The main point to remember is the triggers are "triggered" on an Insert, Update or Delete file (Table) action. Using triggers is similar to putting the code in the `Init` or `Kill` source code locations. One of my programs requires that I increment unique record identifiers manually, instead of using the Clarion autoincrement feature. Now I can add my incrementing code to the dictionary using triggers instead of having to add the code as a separate procedure routine. During certain operations I like to write what is happening to a Log file, so this is also a potential trigger point. You can use triggers to update other files and do various calculations or other lookups. On an Invoice you might want a current amount due that must be calculated after reading part of payment file and summing all payments for that invoice. Certain referential integrity operations or complicated parent-child relationships can also be coded here.

You will find it takes a certain imagination and a little learning time before the full power of triggers will become apparent. Try them; they are quick, easy and powerful.

---

*[Tom Giles](#) is a self-taught, long time CPD 2.1 programmer who started with interpretive BASIC for his business programs. He now uses Clarion 6 for all new projects. When not programming he is out skydiving or flying his homebuilt airplane. Isn't life grand?*

## Reader Comments

[Add a comment](#)

**Tried the Trigger.apps from the examples. it worked fine....**  
**The article failed to mention probably the most important...**  
**Hi there again, the problem i had is in connection with...**

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

**Reborn Free**

**CLARION**  
*online*

## **Understanding Clarion Templates, Part 1**

**by David Harms**

Published 2004-03-26

I don't think it's an understatement to say that templates are the heart and soul of the Clarion development environment. Yes, the Clarion language is highly readable and expressive, and the ABC class library saves developers from reinventing many wheels. Yes, the dictionary editor is massively important. And yes, the database driver system offers a great many advantages. But for my money, nothing says "Clarion" like the template language.

In fact, the templates (and their predecessor, the model files) have been such a huge part of Clarion, ever since Clarion Professional Developer (DOS) 2.0, that they hindered Clarion's adoption of object-oriented technology. Bruce Barrington, Clarion's creator, believed model/template technology had significant advantages over object-oriented programming when it came to improving code maintainability and reusability. Eventually, thanks to David Bayliss and the London Development Centre, OOP did make its way into the Clarion language. Still, I think it's arguable that it's the template language that more than anything defines how most Clarion developers do most of their work.

### **What is the template language, really?**

It's quite possible to develop Clarion applications and know little if

anything about the template language. But as with most programming tools, the more you understand about what's under the hood, the better you can do your job.

The first point about the template language is that it's meaningless without the Application Generator, or AppGen. The AppGen is essentially a template language interpreter. If your programming experience goes back to the DOS days, think about the old BASIC interpreters. You wrote a BASIC program, and fed it to the BASIC interpreter, which (hopefully) produced the desired output, typically a program that interacted with the user to accomplish a needed task. The templates and AppGen function similarly. The AppGen does have some basic hardwired features, such as the report, window and listbox formatters, data dialogs, and the embed editor, but the majority of buttons, entry fields, checkboxes, and other dialogs and prompts you see when you work in the AppGen are defined somewhere by a template. And the actual generation of Clarion source code is almost completely driven by the templates.

And templates, like BASIC programs, are just text files.

Among other things, that means that you can create templates to generate just about anything you like. Years ago, when Java first appeared on the scene, I wrote some rudimentary templates to generate Java code. SoftVelocity, makers of Clarion, have created template sets for ASP and PHP. RADventure's Fenix is a template set that creates ASP.NET applications written in C# and VB.NET. The possibilities are vast (despite being constrained by the AppGen's procedural model, but that's another subject entirely).

## **Template chains and template types**

There are several ways to classify Clarion templates. One way is by template function, or type, another is by file organization, also commonly known as template *chains*.

A chain of templates has a main file with the extension .TPL, and any

number of included files with the extension `.TPW`. This modular structure makes for easier maintenance. You will also quite often see a single template, or set of templates, contained in a single `.TPL` file. Although it probably doesn't make sense to think of a single TPL file as a "chain," there is another important distinction between TPLs and TPWs: templates have to be registered before they can be used (I'll say more about that shortly), and you can only register TPL files. So in order for any template to be available to the AppGen, it has to be either in a TPL, or in a TPW that is included (using the `#INCLUDE` statement) by a TPL.

Templates are often grouped into chains (or just TPL files) by function. These can be application generation templates, utility templates, template wizards, templates from a third party vendor, and so forth. Some of the templates included with Clarion 6 EE, for instance, provide features such as ADO browses, Cybercash and LinkPoint credit card processing, messaging (email and news), business rules, Crystal Reports interfacing, graphing, version resource information, access to the Clarion finance and business statistics libraries, and HTML, PDF, and XML report output.

Besides their physical organization in chains (or single TPLs), templates can also be categorized by type. There are seven basic types of templates:

- Application templates, which control the generation of entire applications. Marked by an `#APPLICATION` statement.
- Code templates, which generate Clarion source code into a single embed point. Marked by a `#CODE` statement.
- Control templates, which are wrappers for Windows controls such as list boxes, windows, etc. Marked by a `#CONTROL` statement.
- Extension templates, which generate Clarion source code into multiple embed points. Marked by an `#EXTENSION` statement.

- Procedure templates, which generate entire procedures, usually in conjunction with one or more of the above templates. Marked by a `#PROCEDURE` statement.
- Module templates, which govern the generation of source modules (which can contain one or more procedures). Marked by a `#MODULE` statement.
- Utility templates, which can be run from the Clarion IDE. These are most often used to report on the application (i.e. the dictionary printout utility), or to create new procedures (the various procedure wizards). Marked by a `#UTILITY` statement.
- Template groups are roughly the equivalent of functions, in the template language. They make it possible to reuse template functionality without having to recreate the same template code over and over. You can pass values to a template group and get values back. Marked by a `#GROUP` statement.

In the coming weeks I'll look in more detail at each of these areas, but for now I want to focus on just application templates, since these are place most developers first encounter Clarion's template technology.

## **Application templates**

Application templates are those templates which are responsible for pulling together all of the other templates (and there may be many) required to create a Clarion application. The Clarion Help defines `#APPLICATION` as marking "the beginning of a source generation control section."

Application templates generate global data (out of the dictionary, and any global embed points), declare global template variables, and provide all of the global option prompts you see in the AppGen for a given application. These templates also control the generation of source code; if a global regenerate is specified by the user, or required because of application changes, these templates generate all

modules, otherwise they generate just the changed modules. Application templates also display the "Generating..." messages. And they do a variety of housekeeping tasks such as adding required database drivers to the project so the application will link properly.

## **Which application chain should I use?**

In Clarion for Windows 1.x and 2, there was only one application template chain. These versions of Clarion used strictly procedural code. That changed with Clarion 4, which introduced the ABC class library (no, there was no Clarion for Windows 3, and as of version 4 the "for Windows" was also dropped as there was no longer any potential for version confusion). Clarion's object-oriented code had radically different template requirements than the procedural code, and it wasn't practical to have one template set capable of creating both kinds of applications, so the ABC template chain was born. The Clarion template chain became informally known as the Legacy template chain, although SoftVelocity seems to favor "Clarion" over "Legacy." I will, however, use the term "Legacy" to refer to these templates and the applications created with them, as I think it has less potential for confusion.

Unfortunately, there is no easy way to migrate a Legacy application to ABC – you can't simply go into the AppGen and change the template chain to ABC. You won't see the ABC chain listed in the available choices, and if you type "ABC" in the Application Template field (at least in C6) you'll get an error message explaining that you cannot change to an incompatible template family.

Conversion was a hot topic when Clarion 4 was released, and all releases from that time on have come with a converter which purports to convert Clarion for Windows 2.003 applications and C4 beta applications to ABC. Look for the program *cnconv.exe* in your bin directory (where *n* is the Clarion release number). The converter is usable for simple applications, but the trickier you get with your Legacy code, the more problems you'll encounter in conversion. For



instance, many Legacy developers got into the habit of replacing functionality in generated Legacy code by placing OMIT directives across embed points to cut out whole sections of code, so they could replace it with their own. Because the generated code changed so radically from Legacy to ABC, those OMIT statements often had wildly different results in the new application, and OMITs that were once inside a procedure body or routine now spanned class methods. Also the converter will not convert your embedded source, and the more use your embed code makes of the generated code, the more trouble you are likely to have.

Most developers I know found it easier to rewrite their applications for ABC, and certainly most new development these days is done with the ABC template chain. But there are still developers who maintain existing Legacy apps, and others who see no need to move to ABC. Some in the ABC camp have lobbied SoftVelocity to drop Legacy support entirely, so that more resources can be devoted to adding new features rather than keeping Legacy feature-compatible with ABC (or mostly so). SoftVelocity's response, as of Clarion 6, has been to modify the Legacy templates so they can begin making use of the ABC class library. And in hindsight, it's easy now to see why SoftVelocity abjured the term "Legacy" – these templates are still alive and well.

Although application templates are a distinct type separate from the other template types (code, extension, control, etc.) they are seldom, if ever, used on their own. Instead, they're the organizing principle behind the use of all other templates in an application. At the end of the Clarion 6 ABCHAIN.TPL file, for instance, you will see the following #INCLUDE directives:

```
#INCLUDE( 'ABWINDOW.TPW' )
#INCLUDE( 'ABASCII.TPW' )
#INCLUDE( 'ABBLDEXP.TPW' )
#INCLUDE( 'ABBLDSHP.TPW' )
#INCLUDE( 'ABBLDWSE.TPW' )
#INCLUDE( 'ABBROWSE.TPW' )
#INCLUDE( 'ABCODE.TPW' )
#INCLUDE( 'ABCONTRL.TPW' )
#INCLUDE( 'ABDROPS.TPW' )
```

```
#INCLUDE( 'ABFILE.TPW' )
#INCLUDE( 'ABGROUP.TPW' )
#INCLUDE( 'ABMODULE.TPW' )
#INCLUDE( 'ABPROCS.TPW' )
#INCLUDE( 'ABPOPUP.TPW' )
#INCLUDE( 'ABOLE.TPW' )
#INCLUDE( 'ABPROGRM.TPW' )
#INCLUDE( 'ABRELTRE.TPW' )
#INCLUDE( 'ABREPORT.TPW' )
#INCLUDE( 'ABUPDATE.TPW' )
#INCLUDE( 'ABFUZZY.TPW' )
#INCLUDE( 'ABUTIL.TPW' )
#INCLUDE( 'cwRTF.TPW' )
#INCLUDE( 'cwHHABC.TPW' )
#INCLUDE( 'CWUtil.tpw' )
#INCLUDE( 'SVFnGrp.TPW' )
#INCLUDE( 'SVUSortOrder.tpw' )
#INCLUDE( 'ENHANCED.TPW' )
#INCLUDE( 'PROCBIND.TPW' )
#INCLUDE( 'ABOOP.TPW' )
#INCLUDE( 'QCENTER.TPW' )
#INCLUDE( 'BrwExt.TPW' )
#INCLUDE( 'ABVCRFRM.TPW' )
#INCLUDE( 'ABOOPU.TPW' )
#INCLUDE( 'REBASEDLL.TPW' )
#INCLUDE( 'UTIL.TPW' )
#INCLUDE( 'ABTHREAD.TPW' )
#INCLUDE( 'ABBLOB.TPW' )
#INCLUDE( 'BLOBSRV.TPW' )
#INCLUDE( 'xmlsprt.tpw' )
#INCLUDE( 'rtfctl.tpw' )
#INCLUDE( 'HelpUtil.tpw' )
```

**That's a lot of templates! In general, the further down that list you go, the more specialized the templates become. The vast majority of Clarion code generation is handled by the templates that deal with the window processing, and with the most common controls, such as reports, browses, menus, and form controls.**

**Next time I'll continue this discussion with closer look at the template language itself.**

---

*[David Harms](#) is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995). His most recent book is [JSP](#).*

*[Servlets, and MySQL](#), published by HungryMinds Inc. (2001).*

## Reader Comments

[Add a comment](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

## Clarion Magazine



## Understanding Clarion Templates, Part 2

by David Harms

Published 2004-03-31

In [Part 1](#) of this series I briefly discussed the function of templates in the Clarion development system, and looked at the seven basic types of templates (application, control, code, extension, procedure, module, utility, and template groups). This week I'll take a closer look at the template language itself.

### The template language

Clarion templates are text files made up of a combination of two programming languages: the Clarion language, and the template language. This, of course, is a recipe for confusion, as well as for productivity.

Template language statements usually, though not always, begin with a # character, and template variable declarations begin with a % character. Template language statements do *not* appear in the generated Clarion source code. Rather, they control the generation of that source.

Here, for example, is a fragment of code from the CodeTPLValidationCode group template:

```
GlobalResponse = RequestCancelled
IF LocalResponse = RequestCompleted
    %ControlUse = %LookupField
```

```
#IF(%ControlEvent='Accepted')
  ELSE
    SELECT(%Control)
    CYCLE

#ENDIF
END
```

You don't need to understand, at least for now, what this code really does. The point is that it shows how a template can (and often does) contain a mix of Clarion language and template statements. The first two lines of code will be generated as is into the Clarion source file. In the third line, the `%ControlUse` and `%LookupField` variables will be replaced by field equates (use variables) as chosen by the developer, in template prompts within the AppGen. The `ELSE` block of code will only be generated if the contents of the `%ControlEvent` variable equals 'Accepted'.

You may also notice that the indentation levels of the Clarion and template source code don't necessarily match up. There's no reason why they should – after all, the logic behind which block of Clarion code should be generated may not have much in common with the logic of the block of Clarion code being generated. Sometimes it's a bit tricky to "see" the Clarion code within a template.

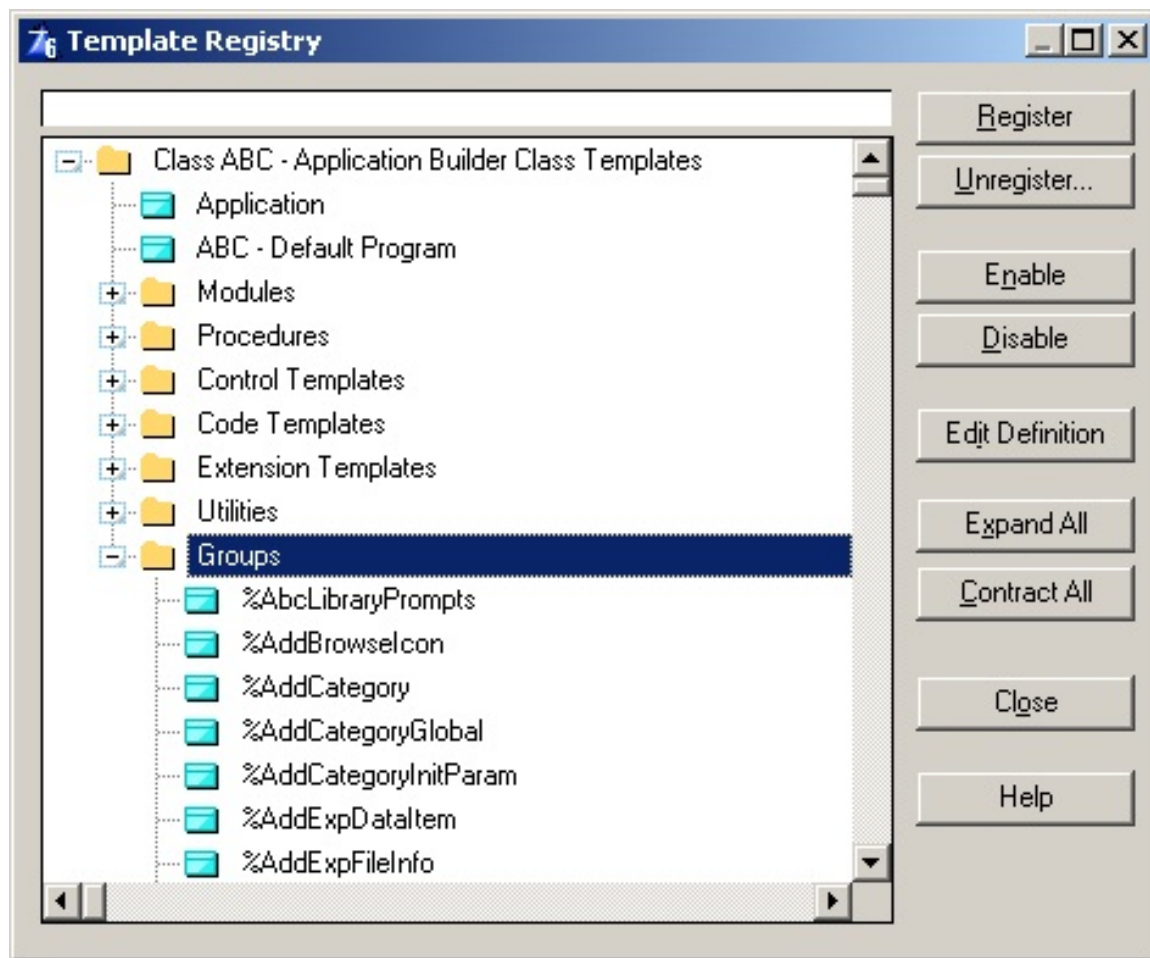
The complexity inherent in the template/Clarion language mix is one of the reasons Clarion almost inevitably grew object-oriented extensions. The Legacy templates (at least prior to C6) generate all of the code for the application, and that means that if you're the person in charge of the templates, and you want to change how some of that code works, you have to be really good at differentiating between the template source and the Clarion source. As of this writing, at least, there is no development tool available that makes this task easy. In the ABC templates, however, most of the functionality is embodied in the all-Clarion ABC class library; the templates function more as wrappers around ABC classes. They don't generate nearly as much actual Clarion source as before.

## Registering templates

To use any template you have to register it in the template registry.

Although the Clarion IDE could, in theory, use the templates directly from disk, a source file isn't the most efficient storage format for this kind of information. The various templates need to be indexed and sorted for quick retrieval, and there's also a certain amount of error checking that has to be done. It's more efficient to do all of this massaging of template data just once, and then have everything ready for the AppGen to use.

To register a template, choose Setup|Template Registry from the Clarion main menu. You'll see the window shown in Figure 1. To register a template click on the Register button and choose one or more TPL files.



**Figure 1 The ABC template chain in the registry**

## Starting with #TEMPLATE

I indicated last week that templates can be organized, and described, in a variety of ways, such as by function, or by their template chain or family. Template chains always begin with a #TEMPLATE statement, such as this

one which is at the beginning of ABCCHAIN.TPL:

```
#TEMPLATE(ABC, 'Application Builder Class Templates'), FAMILY('ABC')
```

The help file for #TEMPLATE gives the following description of this statement's parameters (with my comments added in italics):

```
#TEMPLATE( name, description ) [, PRIVATE ] [, FAMILY( sets ) ]
```

#TEMPLATE	Begins the Template set.
name	The name of the Template set which uniquely identifies it for the Template Registry and Template Language statements. This must be a valid Clarion label.  <i>The name makes it possible for a template in one chain to call a group template (a.k.a. a template function) which is defined in another chain</i>
description	A string constant describing the Template set for the Template Registry and Application Generator.
PRIVATE	Prevents re-generation from the Template Registry.  <i>This is a seldom-used option – the idea, back in the early days of Clarion for Windows, was that you could provide a template registry to someone so they could do development, but they wouldn't be able to generate the original template source from the registry, so you could effectively hide your templates from prying eyes.</i>
FAMILY	Names the other Template sets in which the #TEMPLATE is valid for use.

sets	<p>A string constant containing a comma delimited list of the names of other Template sets in which the #TEMPLATE is valid for use.</p> <p><i>The two usual options here are ABC and CW20 – a template that is available to both, such as the graphing template in C6, will have a family attribute of 'ABC', 'CW20'</i></p>
------	--

There is typically one line of code following the #TEMPLATE statement, and that's a #HELP statement which specifies the default help file associated with the template, for example:

```
#HELP( 'C60HELP.HLP' )
```

From this point on the template is divided up into code sections.

## Template code sections

A template code section is simply a block of code which begins with one of the following template statements:

```
#SYSTEM
#APPLICATION
#PROGRAM
#MODULE
#PROCEDURE
#CONTROL
#CODE
#EXTENSION
#UTILITY
#GROUP
```

The end of a template code section is marked by the end of the file, or by the beginning of another template code section. And although there are different kinds of code sections, the template source they contain can be broken down further into three kinds: prompts, symbol declarations, and code.



## Template prompts

Most templates, including #APPLICATION, #PROCEDURE, #EXTENSION, #UTILITY, #CONTROL, and usually #CODE templates, have prompts which let the developer choose options, select files, fields and controls from pick lists, enter free form text, and so forth. The vast majority of options you have available to you within the application generator are determined directly by the templates.

Template prompts are defined by a #PROMPT statement, as in this example:

```
#PROMPT('&DATA Sections',COLOR),%ColorDataSection,DEFAULT(0000FFH)
```

This prompt lets the user pick a color from the standard color dialog, and save that value in a variable called %ColorDataSection. The available prompt types, from the Clarion help, are as follows:

PROCEDURE	The label of a procedure
FILE	The label of a data file
KEY	The label of a key (can be limited to one file)
COMPONENT	The label of a key component field (can be limited to one key)
FIELD	The label of a file field (can be limited to one file)
EXPR	A multi-field selection box that builds an expression
OPTFIELD	Constant text or the label of a file field
FORMAT	Calls the listbox formatter.
PICTURE	Calls the picture token formatter.
DROP	Creates a droplist of items specified in its parameter

KEYCODE	A keycode or keycode EQUATE
OPTION	Creates a radio button structure
RADIO	Creates a radio button
CHECK	Creates a check box
CONTROL	A window control
FROM	Creates a droplist of items contained in its symbol parameter
EMBED	Allows the user to edit a specified embedded source code point
SPIN	Creates a spin control
TEXT	Creates a text entry control
OPENDIALOG	Calls a standard Windows Open File dialog
SAVEDIALOG	Calls a standard Windows Save File dialog
COLOR	Calls a standard Windows Color dialog

There are a number of techniques you can use in conjunction with #PROMPTS to improve the visual appeal of your templates, include the #TAB and #BOXED statement to create tabs and surrounding boxes, respectively. You can also explicitly set the location of prompts using the AT attribute, and in Clarion 6 you can even [set foreground and background colors](#).

## Template symbols

Templates often declare variables, more accurately called symbols, for the same reasons you declare variables in a source procedure: to store data needed by the template.

To declare a template symbol, you use the `#DECLARE` statement, and you prefix the symbol label with `%` to differentiate it from normal Clarion labels:

```
#DECLARE(%OOPConstruct)
```

If you don't specify a data type, a string is assumed. Possible data types are `LONG`, `REAL`, and `STRING`. Here's a `LONG` variable:

```
#DECLARE(%ByteCount, LONG)
```

Template symbols can also be single valued or multi-valued. A single valued symbol is what you'd expect – it can hold only one value at a time. Multi-valued symbols are like simple queues, containing multiple instances of the symbol. In the following declaration, no type is specified, so this is a multi-valued string, which functions just like a queue with a single string field:

```
#DECLARE(%ClassDeclarations), MULTI
```

You can also link multi-valued symbols, so that one is parent to the other. This is like nested queues. I'll have more to say about multi-valued symbols later on in this series.

Besides user-defined symbols, there are a large number of built-in symbols. These provide information about files, fields, relations, schematics, drivers, views, modules, procedures, windows, reports, formulas, and a number of special-purpose symbols.

## Template code

Not all templates have prompts, and not all have declarations, but it's a rare template that doesn't have any code! The ultimate purpose of a template, after all, is almost always to generate source code, and to do so it needs to contain Clarion code, or template code that generates Clarion code, or some combination of the two. Compared to prompts and declarations, template code is a truly massive subject, and one which I'll begin discussing next time.

## A final word on application templates

In Part 1 of this series I mentioned the #APPLICATION template as the starting point for a template chain. In fact, very few developers will ever deal write an #APPLICATION template, or the related #SYSTEM, #PROGRAM, and #MODULE templates. These templates together manage the generation of Clarion applications, and as long as you're writing Clarion code, there's probably no need for you to reinvent the wheel. If you are creating templates that generate code for another language, such as C++ or Perl, then most likely you will want to investigate these areas more closely.

Instead, most Clarion developers who venture into the template language do so by creating code, extension, utility, and control templates, and those will be the areas I'll focus on in coming articles.

---

*[David Harms](#) is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995). His most recent book is [JSP, Servlets, and MySQL](#), published by HungryMinds Inc. (2001).*

### Reader Comments

[Add a comment](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

## [Clarion Magazine](#)



## **System Tray Popup Windows**

**by Jim Kane**

Published 2004-03-31

Shortly after installing Outlook 2003 I noticed small windows appearing about the tray area when a new email message was received. I found these very useful, so I wanted to see if I could do the same with Clarion. If it had worked on the first try, I probably would have just partly developed it and put it aside in case I ever really needed, but once I failed miserably on the first try, I couldn't stop until I got what I wanted. To spare others the frustration I experienced I thought I'd describe my work here and present the code.

After some thought I developed the following requirements for the tray popup windows, to ensure they were suitably cool:

1. Use slide animation such that they appeared to rise out of the tray area. The window starts off with a zero height and gradually increases in height until full height.
2. Use fade animation after they had displayed for a fixed period of time.
3. Support operation no matter where the tray was located on the desktop (bottom, top, right, or left).
4. Allow closing the tray windows by pressing a close button.
5. Allow moving the windows by pressing the left mouse button and dragging.
6. If the window was clicked on while fading, bring it back to life, i.e. redisplay it.
7. Double clicking should close the tray window and bring up a regular Clarion window showing the full detail for the information

shown in the tray window.

8. I wanted a gradient fill for the window color and some type of frame around the tray window rather than the classic window appearance with a title bar.
9. When the windows opened, they should not be fully overlapped to make them easier to read.
10. I wanted to limit the number of windows open at one time so they didn't take over the screen.
11. I didn't want the windows to be the "always on top" type so that if you were not terribly interested in the popup message, it could be easily ignored.
12. I wanted the option of playing a sound when the windows appeared.

On initial exploration I found two APIs that would be helpful in getting the window appearance I wanted: `AnimateWindow` and `GradientFill`. `AnimateWindow` provides both the fade and slide animation. `GradientFill` provides a way to color the window background with a color gradient rather than with a solid color. The only problem though is neither is supported on Windows 95. As a result I decided to call these functions by address so that my program would still run on, and provide substitute code for, Windows 95. On Windows 95 there are no animation effects – the window appears abruptly, hides abruptly, and the window is a solid color. With these compromises, I could support all Windows operating systems from Windows 95 through 2003.

When I tried to use `AnimateWindow` and `GradientFill` with Clarion windows I didn't have much luck. The built in Clarion window creation and painting code got in the way. My solution was to use the Windows API to create a thread (using the `CreateThread` API call) and a window (using `CreateWindow`) for each popup tray window created. That worked quite nicely.

In the download with this article is `animate.prj` (and `animate.exe`) which should compile in all versions of Clarion from 5 through 6. I recommend you run `animate.exe` to see the popup windows, and then look at `animate.clw` as you read through a description of the important parts of the code below. I think once you see them you'll be hooked too.

I also found a tremendous side benefit to this code. As it turns out, I added the sound effect very early on in the development. I played the sound probably hundreds of time as I developed, compiled, cursed, compiled, ran, cursed, and restarted. What I found it after a few hundred repetitions was that the sound drove my teenage children absolutely crazy. They can listen to rock ten times louder and ten times longer than I can, but playing this one simple sound got their attention. Now when ever they don't want to turn down their music, I threaten to play this tune again in a continuous loop. The house has never been so quiet. If I could only bottle and patent the effect, I'd be rich. Then if I could find another tune that makes babies stop crying, I'd give Bill Gates a run for his position in the world wealth standings.

To use this code, in the application frame's `Event:OpenWindow` embed, START a procedure called `PopupStarter` like this:

```
!this is the thread that starts the tray
! popups - a tray popup factory!
start(PopupStarter, ,address(ShowMsg))
```

The `ShowMsg` procedure is the form or window that is called when the user double clicks on a tray window to show full detail. That procedure should have a prototype like this:

```
ShowMsg procedure(string strMsgID)
```

The single string parameter passed to `ShowMsg` is the unique primary key for the record to be shown in response to the double click. The primary key field can be a long, as Clarion will automatically convert string to long. At the start of `ShowMsg`, typical code, after opening files, would be:

```
PrimaryKeyField = strMsgID
If Access:YourFile.TryFetch(PrimaryKey) then return Level:fatal.
```

The code above will fetch the record to be shown. If `ShowMsg` is a form rather than a window, be sure to set `GlobalResponse` to `ChangeRecord` at the very start of `ThisWindow.Init()`. If you don't want anything to happen on double click, then pass '0' instead of a procedure address



The `PopupStarter` procedure has a timer, and on each `Event:Timer` it checks a global queue called `TrayWndQ` for a message to display. If there is a record in `TrayWndQ`, `PopupStarter` determines if there is room on the screen to display it, and if so, creates and displays the tray popup window. Since `TrayWndQ` is global queue, and I wanted this code to work in Clarion 6, I also created a critical section class called `trayWndQLock` to protect the global queue. Typical code to make a tray popup window appear is:

```
trayWndQLock.EnterCriticalSection()
TrayWndQ.cFrom= 'What ever you want to appear on the from line'
traywndq.cMsg='Message to appear in the tray window'
traywndq.msgid= primary key field associated with this window
add(traywndq,1)
trayWndQLock.LeaveCriticalSection()
```

Since both `trayWndQLock` class and `TrayWndQ` queue are both global, the above can appear anywhere in your code.

This include file contains the `traywndq`, `traywndqlock`, and `PopupStarter` declarations:

```
Include('traywnd.inc')
```

In addition, add `traywnd.clw` and `calla.a` to the external source area of the project tree. `Traywnd.clw` contains the code for `PopupStarter`, and `calla.a` helps with the call by address for Windows 95 support. The call by address scheme used and `calla.a` are described in my article [Call By Address, STARTing By Address](#).

`PopupStarter` starts out by determining how many dots per inch the screen uses. All the pixel values used in the design of the popup windows assume 96 dpi. If the actual screen resolution is not 96 dpi then it scales all values appropriately for what ever the run-time screen resolution is. This allows the windows to work even if a user has selected large fonts, or has an unusual display device. `PopupStarter` also checks to see if the number of popup tray windows allowed to be on the screen at once will actually fit. If not, it scales back the maximum number of windows it will open.

PopupStarter tracks all the popup tray windows it has already started through the use of a global array declared in traywnd.clw (which uses a critical section class called critscl for thread protection):

```
!global data - use critscl for access
eMaxTrayWnds equate(5) !if 5 or more popups are shown and you
                    !try to start a 6th, it is not started.
Event:FirstSlot equate(0A200H) !events for the double click handler
Event:lastslot equate(Event:FirstSlot+eMaxTrayWnds-1)
Glo:TrayWndMgr group,dim(eMaxTrayWnds)
threadid      long
lpdata        long
hthread       long
end
```

In my sample code, I elected to have no more than five tray popup windows at one time. The eMaxTrayWnds equate declares this limit. For each window, the Glo:TrayWndMgr array stores the threadID, hThread or thread handle, and a pointer to thread-specific data, lpData. If the threadID is 0 the array entry is assumed to be zero, and can be used for a new window.

lpData points to a rather large data group with all the data the API thread needs to display the popup window

```
tlsType      Group,type
hwnd         long !hwnd of the popup window
parenthandle long !handle of the dummy window
              ! of PopupStarter
slotidx      long !1...eMaxTrayWnds
PopupStarterThread long !clarion thread # for PopupStarter
useslide     byte
xoffset      long !offset from edge of the screen
yoffset      long
r            like(recttype) !rect for drawing the edge
Gradient_Rect like(GRADIENT_RECTType)
vertex       like(vertextype)
xbutton      like(recttype) !close button rect
dpix         long !dpix for font calcs
MaxScreenX   long !screen size in X
MaxScreenY   long !Screen size in Y
hfont        long,dim(eMaxFont)
xpos         long !x,y pos of the window
ypos         long
mouseleftdown long !set to true when the mouse is
              ! down regardless of why
```

```

mousedrag                long  !set to true after mouse down for 1
                          ! set - beginning of a drag
MouseDownX               long
MouseDownY               long
buttondepressed          byte !set to true when button depressed
                          ! on mousedown
cFrom                    cstring(50)
cMsg                     cstring(400)
MsgID                    long !primary key of message
iconresourcename         cstring(260) !must be linked into the app (not dll)
iconsize                 long
!Windows 95 compatibility
lpGradientfill           long
lpAnimatewindow          long
end

```

To customize the appearance of tray windows, modify both the `traywndq` and the `tls` group above to hold what ever you need for the window appearance you want. In addition to the information passed to the `PopupStarter` in the global queue, there are quite a number of equates that determine various aspects of the tray popup windows appearance and actions. To locate these equates that determine the window appearance for all tray windows, search the source of `traywnd.clw` for this comment:

```

"!----- Start of equates for appearance and
function -----!"

```

All the items in that section are user preference so have at it and change them to get the desired appearance. One item to note is there is one equate for each font to be used, along with an equate of the total number of fonts used on the window. If the window design requires more than three fonts, change the `eMaxFont` equate value to the number of fonts you require, and create a new equate for each font to be used.

When `PopupStarter` finds a record in the `traywndq` and an empty spot in the `glo:trayWndMgr` array, it copies the data in the `traywndq` record to the `tls` group allocated by the `Clarion New()` function, along with other data contained in the series of equates described above that pertain to all windows. It then calls the API `CreateThread` function to start a new API thread, which immediately calls the `ShowPopup` procedure. Once the

API thread is created, it deletes the record from the global queue, `trywndq`.

The `Showpopup` procedure is responsible for creating the tray popup window and starting a message loop. Before doing anything else though, if the code is running under Clarion 6, the API thread has to be registered with the Clarion runtime:

```
compile('***', _C60_)
AttachThreadToClarion(0)
!***
```

Once that is done, `ShowPopup` calls `ShAppBarMessage()` to find out where the tray area is, and to determine the xy position at which the new window should appear. Once `ShowPopup` has the xy position, it calls the `CreateWindow` API function to create the window. At this point the window is still hidden. Next, all the fonts needed to paint the window are created. The `AddFont` procedure uses normal Clarion color and font style equates to create API fonts. A `FontID` is passed to the `AddFont` procedure, and the font is subsequently referenced by this `FontID`. These fonts are created and stored in the per thread data so that there is no danger two threads would try to use one at the same time. A timer is then created to control the lifetime of the new window. When the timer fires, a `wm_timer` message is generated and the window is closed.

If during the startup of `PopupStarter` the address for the `AnimateWindow` API function was found, indicating this is not a Windows 95 machine, then the `AnimateWindow` API is called to provide slide animation and display the window. On Windows 95, `ShowWindow` is used instead to display the window. Lastly a very simple message loop is started for the window. A message loop is very similar to a Clarion accept loop, but without any Clarion-specific behavior. When the window closes, the `procret` routine is called to free the font objects and remove the window from the `glo:traywndmgr` array.

All the Windows messages for the window created by `ShowPopup` are processed in the `SubClassFunc1` procedure. Of most interest for customization are the `wm_paint` event and the closely related `wm_print`

and `wm_printclient` events, which are called by `AnimateWindow`. Upon receiving one of these events, the first order of business is to paint the window background. The `GradientFill` API call is used for that unless the function is not found, as would be the case on Windows 95. When `GradientFill` is not available, the default window background is used. Next the `DrawEdge` API function is called to draw the frame around the window edge. With that done, the `TakePaint` procedure is called. `TakePaint` receives a pointer to the `tls` data and the `hdc` or windows device context. `TakePaint` displays an icon if an `IconResourceName` value was supplied. Next, `TakePaint` uses the fonts created earlier to display text on the window.

I've provided two different procedures to make it easy to display text. `PrintString` simply displays the string passed to it using the specified `FontID`. As I mentioned earlier, the fonts are all created before starting the thread, and each font is referenced by its `FontID`.

`DrawString` is a bit more complicated. It can display multi-line text, and if the text is too long it truncates the text and adds an elipsis (...) at the end. Because it can display multi-line text, it takes not just an `xy` position as does `PrintString`, but also a height and width. Also note that all the pixel coordinates for positioning the text are transformed by the `scalex` or `scaley` procedures. This is a part of the run-time scaling used so the window will display reasonably at other than the standard 96 dpi resolution I used during design. Note also the `cstrings` containing the text to display come from the `tls` data group. The strings got into that group from the `TraywndQ`. So to modify the appearance of the window and add or subtract fields in the display, modify the `traywndq` so the required data is passed in, and also change the code just before the `CreateThread` API call where the `traywndq` fields are copied to the `tls` data and the `tls` group itself. By calling customized combinations of `PrintString`, `DrawString`, or the icon drawing code, you can obtain any desired window layout.

If the tray window is double clicked the `wm_lbuttondblclk` message is generated. Timers are killed and a message is posted to `PopupStarter`. `PopupStarter` determines which window was double clicked (based on

the value of the event generated) and starts a form procedure. Once the form procedure is started, `PopupStarter` posts a `wm_destory` message to the tray popup window to close it.

When either the timer controlling the tray popup window's life fires, or the close button is pushed and released, the `wm_close` message is generated. In `WM_Close` timers no longer needed are killed, and the animation, typically fade animation, is used to begin hiding the window. After the animation is completed, `GetInputState()` is called to see if a user clicked on the window while it was fading out. If so the window is brought back to life and a new timer started. If not, the `wm_destory` message is posted. The `WM_Destory` message handler calls `PostQuitMessage()`, and that causes the message loop in `ShowPopup` to end. When the loop ends, `ShowPopup` cleans up memory in the `procret` routine, and then the `ShowPopup` procedure and the API thread created by `PopupStarter` end as well. This completes the life cycle of the popup window.

When your application closes, the Clarion runtime library sends a `event:closewindow` to `PopupStarter`. `PopupStarter` frees the `traywndQ` so no new windows are started. It then loops thru the `glo:Traywndmgr` global array protected by the `critscl` critical section class to find any popup tray windows still running. When they are found, it posts a `wm_destory` message to them to close them. It keeps looping until all the tray windows are closed. It then does a little cleanup and terminates.

While for everyday programming the ease of use of Clarion is great, sometimes it's fun and productive to use the API alternatives to get special effects. I hope you will find this code as interesting and useful as I have.

[Download the source](#)

---

*[Jim Kane](#) was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard*

*University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and after graduating served in the US Air Force. He is now retired from the Air Force and writing software for [ProDoc Inc.](#), developer of legal document automation systems. In his spare time, he runs a computer consulting service, Productive Software Solutions. He is married to the former Jane Callahan of Cando, North Dakota. Jim and Jane have two children, Thomas and Amy.*

## Reader Comments

[Add a comment](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.

**Reborn Free****CLARION**  
*online*

## Compiled Reports From Report Writer

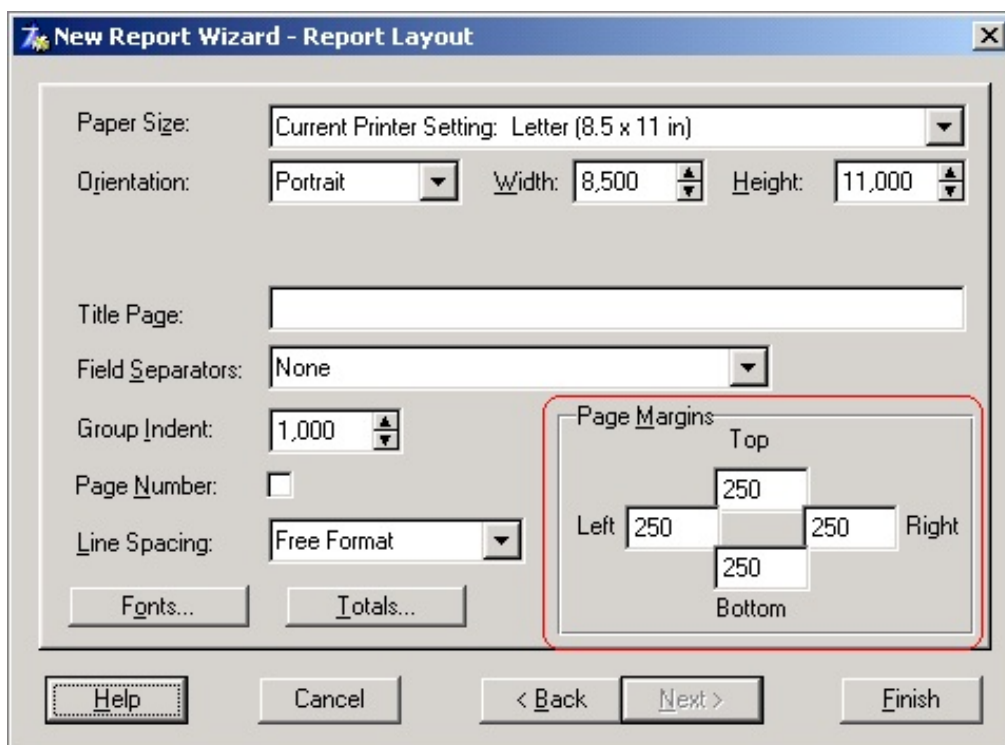
by **Henry Plotkin**

Published 2004-03-31

As far as I am concerned, the best thing about Clarion Report Writer is the ability to set up the finished page without having to adjust the position of each individual band (see Figure 1).

The next best thing about Report Writer is that report testing is much faster. A Report Writer does not require a compile-link cycle. It does not require minimizing Clarion, starting the application and navigating to the report.

Not having to recompile, etc., can save me one or two (full) minutes in testing a report. Because I get very impatient when making changes, this minute or two seems to drag on forever.



**Figure 1. Report Writer page configuration**



What would be ideal is to use Report Writer to create and test reports and, when the report is complete (or nearly so), import it into a standard, compiled report procedure.

I am going to show you how to do just that, import a Report Writer report into a standard .APP with only a little work.

## The TXR

In the accompanying sample application (download at the end of this article), I have included a C5.5 Report Writer library. The first report lists customers and their total purchases. There is a grand total band with the total purchases for all customers.

The .TXR file created for this report (all Report Writer reports are stored in text files with the extension "TXR") is very instructive:

```
[LIBRARY]
VERSION(4001)
ENDUSER('off')
OPTION(0)
[REPORTS]
Report1 REPORT, FONT('Arial',10), PRE(Report1), THOUS, AT(250,912,8000,9838), &
MARGINS(250,235,250,250) !Customer List
    HEADER, FONT('Times New Roman',10,0,700), AT(250,235,8000,677)
        STRING('Customer Number'), AT(50,350)
        STRING('Last Name'), AT(1375,350)
        STRING('First Name'), AT(2888,350)
        STRING('Total Sales'), AT(4610,350)
        LINE, LINEWIDTH(15), AT(0,618,8000,0)
        STRING('Customer Sales Report'), AT(3046,67)
    END
__REPORT__ BREAK
__TOTALS__ BREAK
Detail1 DETAIL, FONT('Arial',8,0), AT(0,0,,253)
    STRING(@n-14), LEFT, TRN, USE(CUS:CustomerNumber), AT(50,0,1125)
    STRING(@s20), USE(CUS:LastName), AT(1375,0)
    STRING(@s20), USE(CUS:FirstName), AT(2888,0)
    STRING(@n-10.2), USE(CUS:TotalSales), AT(4401,5,844)
    END
    FOOTER, AT(0,0,,1000)
        STRING('Total Sales:'), AT(3682,189)
        STRING(@n-10.2), USE(GrandTotal), AT(4528,189)
    END
    END
    END
GrandTotal TOTAL(@n-10.2), SUM, USE(CUS:TotalSales)
    END
[FILES]
Customers FILE, PRE(CUS), DRIVER('TOPSPEED',, 'C55tps', 'TPS'), CREATE
CustNbrKey KEY(CUS:CustomerNumber), NOCASE, OPT
CusNameKey KEY(CUS:LastName, CUS:FirstName, CUS:CustomerNumber), DUP, NOCASE, OPT
__Record RECORD
CustomerNumber LONG, PICTURE(@n-14), PROMPT('Customer Number:'), &
```

```

HEADER('Customer Number')
LastName      STRING(20),PICTURE(@s20),PROMPT('Last Name:'),HEADER('Last Name')
FirstName     STRING(20),PICTURE(@s20),PROMPT('First Name:'),¿
HEADER('First Name')
TotalSales    DECIMAL(7,2),PICTURE(@n-10.2),PROMPT('Total ¿ Sales:'),HEADER('Total Sales')
              END
              END
[RELATIONS]
[REPORTVIEWS]
Report1 VIEW(Customers),ORDER('UPPER(CUS:LastName),UPPER(CUS:FirstName),¿
CUS:CustomerNumber'),KEY(CUS:CusNameKey)
              END
[SOURCES]
              DICTIONARY,VERSION('1.0'),DATE(74229),TIME(3200300),NAME('C:\C55\REPORT~1\¿
SAMPLE.DCT')
[SEARCHPATHS]
              PATH('Customer.TPS','C:\C55\REPORT~1\')
[EXTERNALS]
Notice that it is composed of a number of sections, like an INI file. These sections are:
[LIBRARY]
[REPORTS]
[FILES]
[RELATIONS]
[REPORTVIEWS]
[SOURCES]
[SEARCHPATHS]
[EXTERNALS]

```

The content of several of these sections is entirely self-explanatory. In [SOURCES], the source dictionary is named (this is what allows Report Writer to load the dictionary and detect any changes in it – at least, when you run Report Writer from the Clarion IDE). [FILES] contains all the files (including global variables) from the dictionary, [REPORTVIEWS] contains views used in reports, "indexed" by report and [RELATIONS] contains dictionary defined file relations.

The most interesting section is [REPORTS]. This section contains all the reports defined in the library. The Label of each report is the name given to the report when it is created. However, examining the [REPORTS] section, in isolation from the rest of the file should give one a sense of *déjà vu* – all over again.

As I examine the code in the [REPORTS] section, I find that I cannot distinguish it from the code generated by the Report Formatter in the IDE. There is a REPORT declaration, a prefix declaration, headers, details, breaks and footers, exactly as I would see them if I looked at the Report Formatter's code.

This makes me wonder whether it isn't possible to copy the report layout and paste it into the Report Formatter. The ellipsis button beside the report button on the procedure properties dialog makes this prospect very tantalizing, very tantalizing indeed.

## "Importing" from Report Writer

Of course, before copying and pasting, it is wise to press the Tables button and complete the file schematic for the procedure. *Then* you can open the report code and paste the report structure.

However, on closing the embed, I immediately got a "Syntax error reading WINDOW or REPORT." Not only can't the IDE tell me whether the offender is a window or a report, it doesn't go to the line containing the error.

Not to worry, on examination, it is clear that MARGINS is not a recognized attribute in the Report Formatter. Here's the problem code noted in boldface:

```
Report1 REPORT, FONT('Arial', 10), PRE(Report1), |
  THOUS, AT(250, 912, 8000, 9839, MARGINS(250, 250, 250, 250))
```

Removing the MARGINS attribute solves that problem. But, it brings up another, which is the total declaration in the report structure:

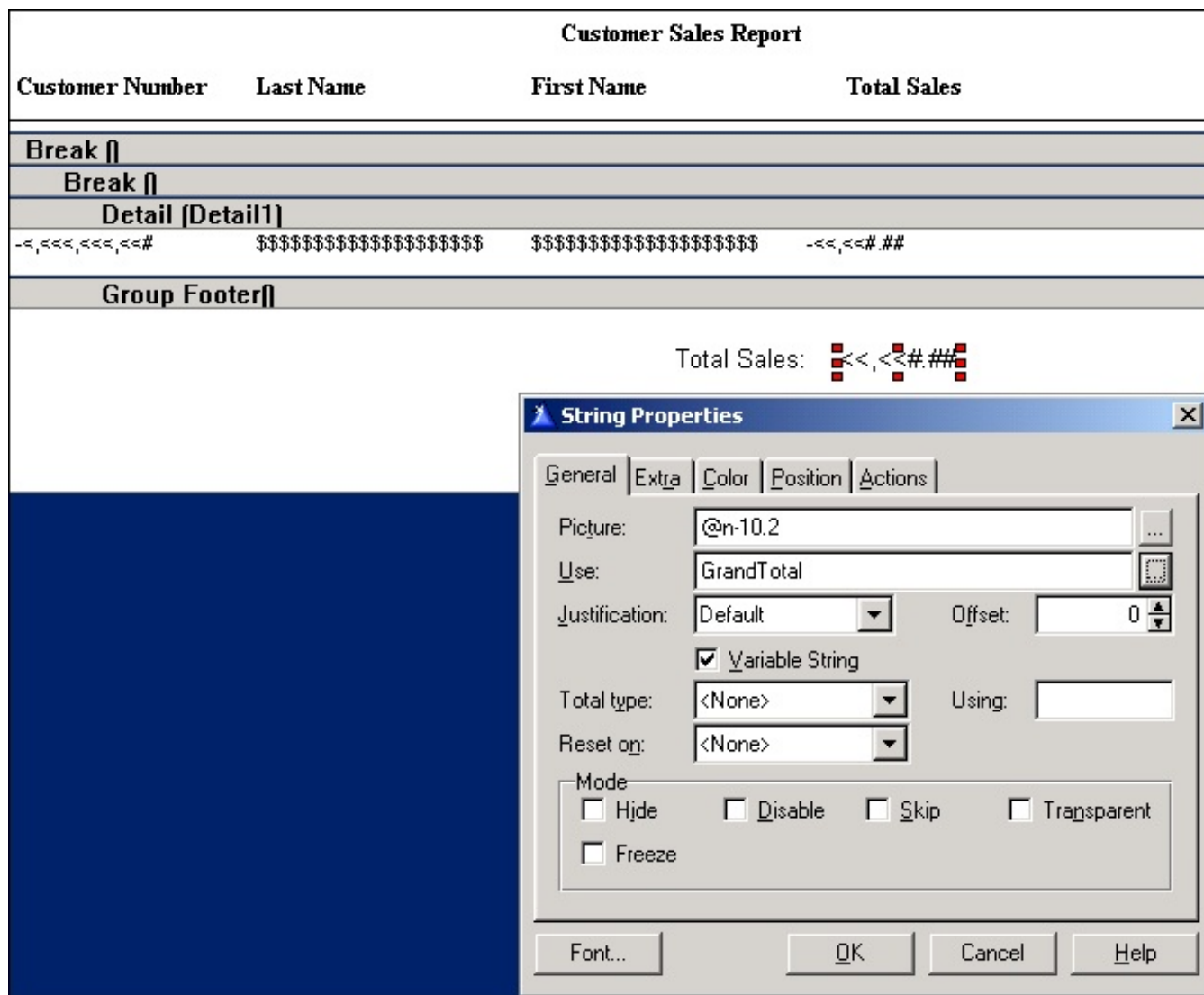
```
GrandTotal TOTAL(@n-10.2), SUM, USE(CUS:TotalSales)
```

The variable GrandTotal is not recognized. Neither is the attribute: TOTAL(@n-10.2). Of course, I could declare GrandTotal and I could change the line to something like:

```
Grandtotal      String(@n-10.2), SUM, USE(CUS:TotalSales)
```

and return to this embed and re-paste the code.

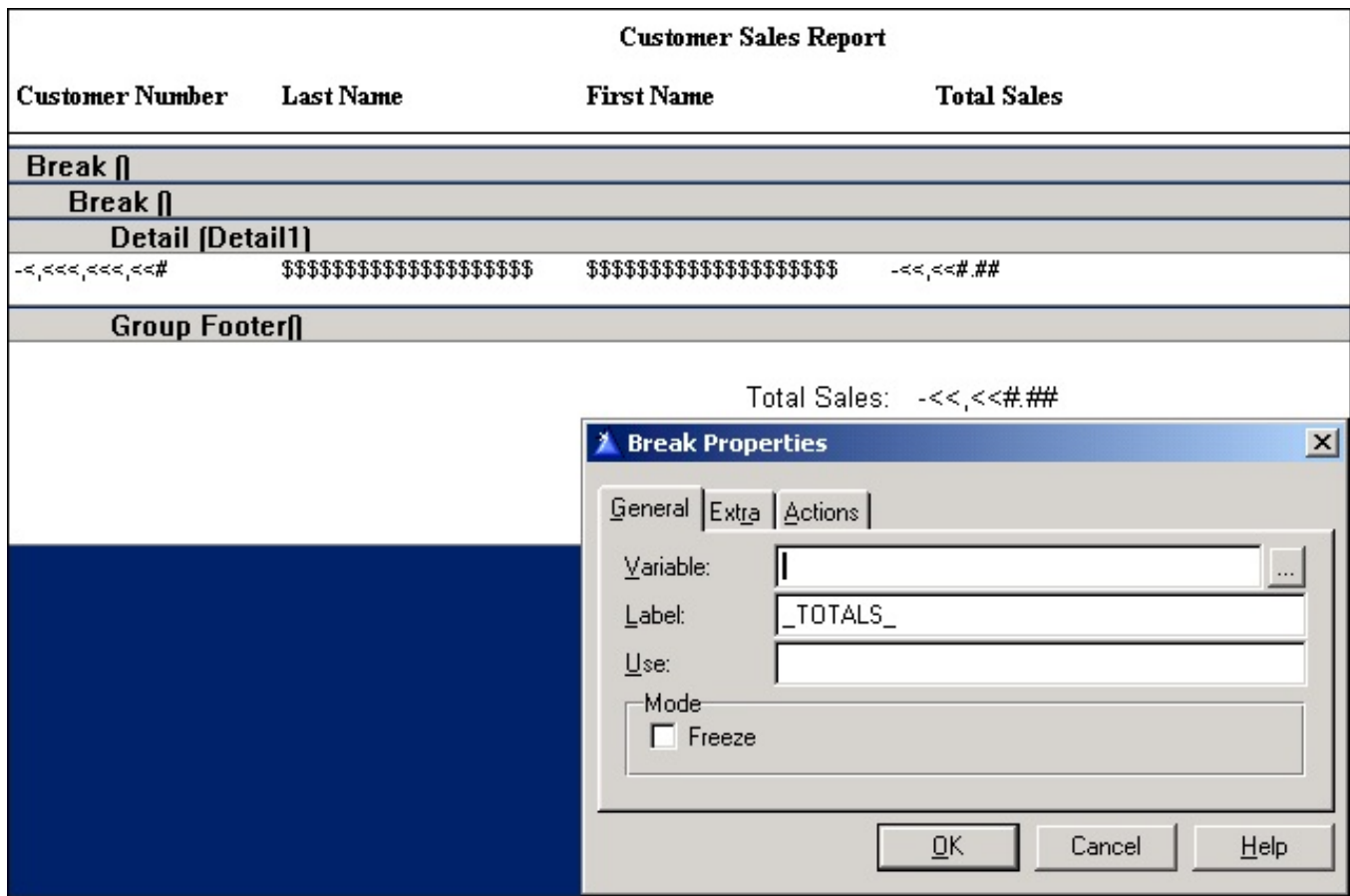
Or, I could delete the offending line and create the variable. Some action is necessary because the report still contains a band with this variable (see Figure 2).



**Figure 2. Total variable remains on report even after deletion**

In the end, I just incremented the variable in TakeRecord and created a grand totals band as described in the on line help.

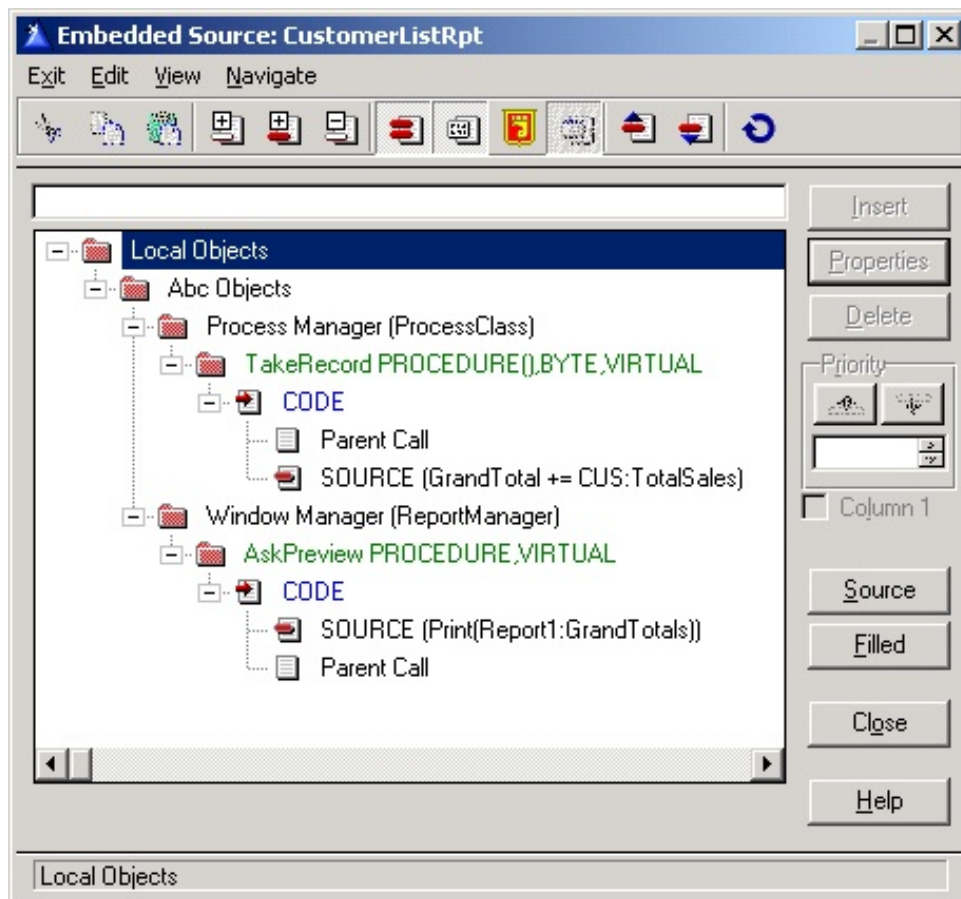
This turns out to be a good idea. If you examine Figure 2, above, you will notice two Breaks. Neither has a Break Variable; neither has a Use variable. These bands appear totally inexplicable.



**Figure 3. Properties of the "mystery" breaks**

But, delete them and the Group Footer, containing the GrandTotal variable disappears. As I said, it was a great stroke of luck that I decided that grand totals weren't so hard to implement without Report Writer's help.

In the end, I end up with two standard embeds. I accumulate the totals in TakeRecord and I print my GrandTotals band in AskPreview (Figure 4).



**Figure 4. The completed embed tree**

But...

Does the finished report work (you know, after compilation, navigating and all that other stuff)?

Yes, it does.

[Download the source](#)

---

*"hp" in fact prefers Hewlett-Packard printers but will use whatever is available. Born in New York City, hp is a self-taught Clarion developer doing a substantial amount of work for hospital gift shops.*

## Reader Comments

[Add a comment](#)

Copyright © 1999-2003 by [CoveComm Inc.](#) All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited.