

Clarion Magazine


online sales and delivery
for your applications & tools

Developer **PLUS**

Please note that Clarion Magazine is temporarily publishing three issues per month rather than four so we can devote more time to our upcoming [book series](#). All subscriptions are automatically extended.



[PDF For December, 2003](#)

Articles: 

All Clarion Magazine articles for December, 2003 in PDF format.

News: 

Posted Monday, January 05, 2004

News

[Accessing Version Resource Information](#)

It's often useful to know the version of a particular EXE or DLL your customer is using, and Windows EXEs and DLLs do have a standard format for version information. There are tools readily available to put this information in your Clarion apps, and now Brian McGinnis demonstrates how to extract that information at runtime.

Posted Tuesday, January 13, 2004

[RPM6 for Legacy\(Clarion\)
Available Now](#)

[FUSE Products C6
Compatible](#)

[OutlookFUSE 2.0](#)

[XMLFUSE 2.0](#)

[ThinkData 25% Off Sale
Through Feb 1](#)

[EasyAutoEntry 1.03](#)

[xAppWallpaper v1.5](#)

[Clarion Training Bookings
In Centurion, Pretoria](#)

[BoxSoft Super Stuff 6.10](#)

[ClarionPost.com
Temporarily Unavailable](#)

[Porting to the C6 Threading Model: The AutoLog Example](#)

Clarion C6 has gone gold and so it's time to start getting serious about porting applications. In this series of articles Carl Barnes looks at porting some of the C55 example applications to C6.

Posted Friday, January 16, 2004

[Weekly PDF for January 11-17, 2003](#)

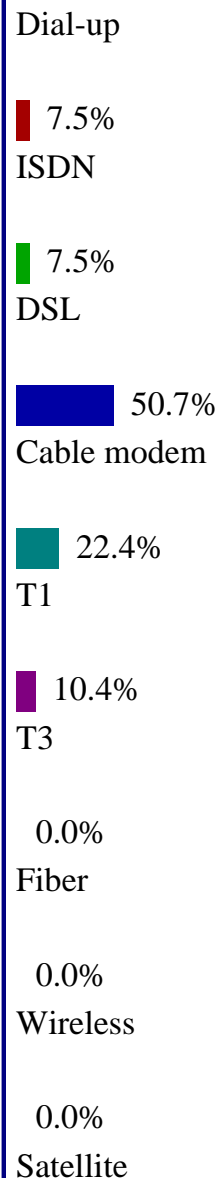
All articles for January 11-17, 2004 in PDF format.

Posted Tuesday, January 20, 2004

[Porting to the C6 Threading Model: The AutoLog Example Part 2](#)

SURVEY

What kind of Internet connection do you have?



Clarion C6 has gone gold and so it's time to start getting serious about porting applications. In this series of articles Carl Barnes looks at porting some of the C55 example applications to C6. Part 2.

Posted Thursday, January 22, 2004

[Addressing Arrays On A Window](#)

Steve Parker loathes, hates, and despise arrays. But sometimes, like it or not, he has to use them. Here's what he learned about using arrays on windows, and what you need to know

Posted Friday, January 23, 2004

[No Issue This Week](#)

There will be no issue this week as we're hunkered down putting the finishing touches on the Tips & Techniques book. As usual all subscriptions are automatically extended.

Posted Tuesday, January 27, 2004

[Tips & Techniques Book - We're Almost There!](#)

We've been aiming for an end-of-January release for the book, and it should be close. We have one or two cover tweaks to finish, and then some final testing of the fulfillment process. The pre-order response has been terrific, and we're very excited about Clarion Magazine's first-ever printed book!

Posted Wednesday, January 28, 2004

[PDF For January, 2004](#)

All Clarion Magazine articles for January, 2004 in PDF format.

Posted Wednesday, January 28, 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.

[Ready-to-go New Products](#)

[SelfService v1.11](#)

[ZipApp 1.2b](#)

[Richard Rose Heads Clarion User Group](#)

[Web Site Reviews](#)

[Redirection Test Utility Program](#)

[Taboga Software Holiday Schedule](#)

[Training for Gauteng, SA](#)

[Fenix Customer Quotes](#)

[EasyAutoEntry 1.02](#)

[ClarioNET Developers Demo System](#)

[BST,BoTpl C6 Patch](#)

[Free Point Of Sale Software Written In Clarion](#)

[ClarioNET 1.4 Release](#)

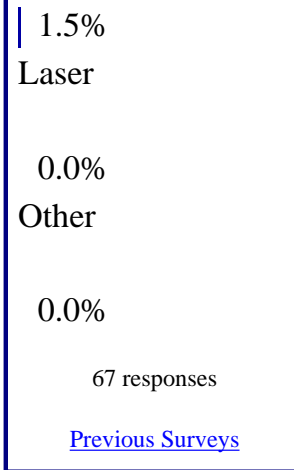
[Clarion Jobs Page Updated](#)

[Firebird ODBC Beta 2](#)

[Ready-to-go Web Templates](#)

[Big gCal Sale](#)

[In-Memory Database Driver](#)



One Year Ago In CM

[Book Review: Programming PHP](#)

[A Calculator Class And Template](#)

[Debug De Program With Debugger](#)

Two Years Ago In CM

[Weekly PDF for January 20-26, 2002](#)

[First Field, Required Field](#)

[Weekly PDF for January 13-19, 2002](#)

Three Years Ago In CM

[Data Is Executed Policy - A Case Study](#)

[Look Ma, No Keys!](#)

[Sending Clarion](#)

[Information](#)

[Reports as Email
Attachments \(Part 2\)](#)

[CHT Website Redesign
Completed](#)

**Four Years Ago In
CM**

[Soggy Systems Products](#)

[Clarion News -
January 2000](#)

[BST Ver 2.6](#)

[Free Popup Alarm](#)

[The Cranky
Programmer](#)

[BoxSoft Super Invoice 6.04
Released](#)

[Skeleton Basics III:
Colors and
Backgrounds](#)

[BoxSoft Super Import-
Export 6.03 Released](#)

[ADDA 1.0.0 Released](#)

[Clarion Resources And
News Page](#)

[News From 1st Logo Design](#)

[Clarion Training In
Capetown](#)

[Vote For etc 2004
Sessions/Classes](#)

[ClarioNET User
Registration](#)

[1st Logo Design Price
Increase](#)

[Clarion Developers
Challenge Grand Prize
Winners Announced](#)

[Clarion Developers
Challenge Week 17 Winners](#)

[Announced](#)

[Search the news archive](#)

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



Accessing Version Resource Information

by **Brian McGinnis**

Published 2004-01-13

It's often useful to know the version of a particular EXE or DLL your customer is using, and Windows EXEs and DLLs do have a standard format for version information. You can add this information using Larry Sand's Version Information Resource Compiler available for download here, the Locus templates, and CWRES from Innovative Software Creations. Now Clarion 6 also includes this capability with the cwVersion template.

The question follows, how can you access this information from inside your program for display on a title bar or inside of an "About" box? There are a few approaches you could take to solve this problem. You *could* write an additional template that would extract these values from the original template and put them into string constants. On the other hand, a technique that works well and is perhaps a bit more flexible, is to simply access this version resource information directly from inside your program using Windows API calls. The latter approach is the subject of this article.

Version information

If you've ever looked at a file's properties from Explorer and clicked on the Version tab, you might see something similar to Figure 1.

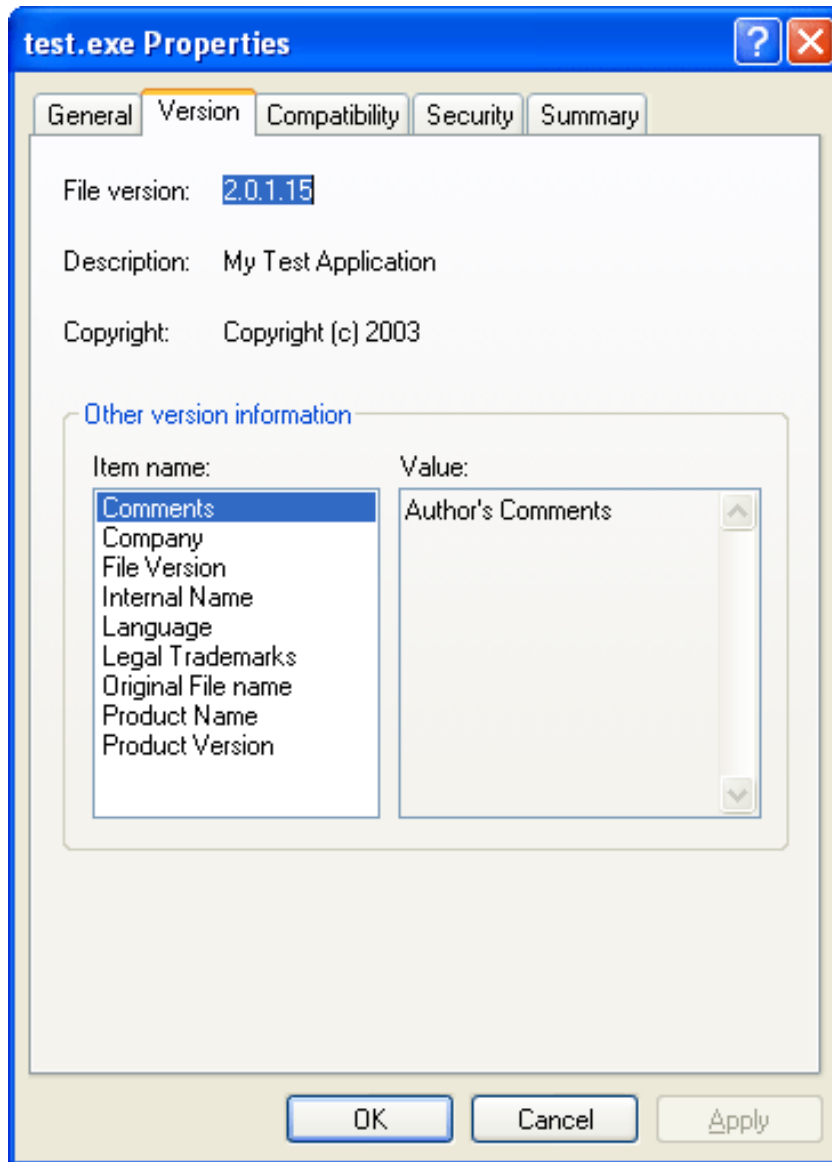


Figure 1. Application properties in Windows XP

Note that the templates I mentioned earlier, including the one provided by SoftVelocity in Clarion 6, provide a means of automatically generating build numbers with each compile. This is handy when you're in the position of having to provide multiple updates to your client the same day. Each build of your application retains the major and minor version numbers you've supplied on the template, but receive an ever incrementing build number with each compilation.

You can use the following Windows API calls to retrieve the version information:

GetFileVersionInfoSize	Determines if the OS can get version resource info for a specified file, and how big it is if it can.
GetFileVersionInfo	Retrieves the version resource information into a buffer.
VerQueryValue	Retrieves specific version information from the buffer.

A better idea would be to design a small class that includes methods to access the basic version strings commonly found in the version resource info, along with a template wrapper to make using it that much simpler. Indeed, accompanying this article is a class and template that does exactly that and works with either the Clarion or the ABC template chains.

A closer look

Execute the API calls in the order presented above to begin retrieving resource info.

`GetFileVersionInfoSize` is a pretty straightforward call and can be prototyped a few different, but equally acceptable ways:

Style 1:

```
bdGetFileVersionInfoSize( *CSTRING, *ULONG ), ULONG, |
    PASCAL, RAW, NAME( 'GetfileVersionInfoSizeA' )
```

Style 2:

```
bdGetFileVersionInfoSize( LONG, LONG ), LONG, PASCAL, |
    RAW, NAME( 'GetfileVersionInfoSizeA' )
```

The function takes two pointers: one to the pathname of the file of interest, and another to a variable that the function sets to zero. If version information is available, it returns the size, in bytes, of that information.

If you use Style 1, your calling code would look similar to this:

```
FName          CSTRING( 256 )
Dummy          ULONG
BufSize       ULONG
CODE
Fname = 'C:\SamplePath\SampleFile.exe'
BufSize = bdGetFileVersionInfoSize( Fname, Dummy )
```

On the other hand, Style 2 would look more like this:

```
FName          CSTRING( 256 )
Dummy          ULONG
BufSize       ULONG
CODE
Fname = 'C:\SamplePath\SampleFile.exe'
BufSize = bdGetFileVersionInfoSize( ADDRESS( Fname ), ADDRESS( Dummy ) )
```

Note the change in prototypes from `ULONG` to `LONG`. This is because the `ADDRESS()` function returns a `LONG`.

The next step is to go ahead and actually retrieve the data into a buffer. This is accomplished using a call to `GetFileVersionInfo()`. The listing below is taken directly from the class and shows the first two API

calls being made using the second style:

```
bdVersionInfo.GetVersionInfoData PROCEDURE() !,BYTE,PRIVATE
! Attempts to fetch version info on file. Once it has
! this data, queries can be made against it.
! Returns FALSE if successful.
```

Dummy ULONG

```
CODE
!Fetch the size of version info data
SELF.BufSize = bdGetFileVersionInfoSize(ADDRESS(SELF.FN),ADDRESS(Dummy))
IF SELF.BufSize <= 0 THEN RETURN TRUE.
SELF.Buffer &= NEW(CSTRING(SELF.BufSize+1))
!Fetch version info into SELF.Buffer
IF ~bdGetFileVersionInfo(ADDRESS(SELF.FN),Dummy,|
    SELF.BufSize,ADDRESS(SELF.Buffer)) THEN RETURN TRUE.
RETURN FALSE
```

The first call attempts to obtain the size of the version resource data and creates a CSTRING to hold it.

The second call attempts to load this data into the SELF.Buffer property of the class. It takes a pointer to the pathname of the file of interest, a second parameter that is ignored, the size of the buffer, and finally a pointer to the buffer itself. If the call succeeds, the return value is non-zero.

The final step in this process is using VerQueryValue to obtain the actual version strings. This is where things can get a bit tricky. First of all, there are two ways you can call this function. The first simply returns a pointer to a structure known as VS_FIXEDFILEINFO that contains basic version resource information. VS_FIXEDFILEINFO does not contain any user-defined strings, nor company name, comments, or file description. The second yields all of the information, but in as many languages and code pages as the author has implemented. This is the approach I've taken.

Enumerating code pages and languages

Normally, the first step is to use VerQueryValue to enumerate which languages and code pages are available in the resource, then use that newly obtained information to retrieve the appropriate strings.

In this case, I know what languages and code pages are in the resource because I put them there, so the enumeration step can be skipped. I assume the same is true for any developer. Therefore the class implements the call to VerQueryValue as follows:

```
bdVersionInfo.GetInfo PROCEDURE(STRING xStrName) !,STRING
! Returns value for a given version string.
! Returns string containing value for matching
! string, empty if not found
```

```
SubBlock CSTRING(128)
VersionPtr LONG
VersionSize LONG
BufInfo BYTE,DIM(1024)
```



```

RVal CSTRING(128)

CODE
IF xStrName = '' THEN RETURN ''
IF ~SELF.Inited
    IF SELF.Init() THEN RETURN TRUE
END
SubBlock = '\StringFileInfo\' & SELF.LangID |
    & SELF.CodePageID & '\ ' & CLIP(xStrName)
VersionPtr = ADDRESS(BufInfo)
IF bdVerQueryValue(ADDRESS(SELF.Buffer),ADDRESS(SubBlock), |
    ADDRESS(VersionPtr),ADDRESS(VersionSize))
    bdMemCpy(ADDRESS(RVal),VersionPtr,128)
END
RETURN Rval

```

This method is handed a version string to search for and returns the matching value if found. If you do not set the LangID and CodePageID properties, the default values of multilingual and US English are used.

VerQueryValue takes a pointer to the buffer that contains the version resource information obtained from the previous call, a pointer to the resource string whose value is to be obtained, a pointer to a variable that receives a pointer to the requested information within the buffer, and finally a pointer to a variable that receives the length of the information in characters.

Essentially, VerQueryValue is indicating where the value string is at in the buffer obtained from an earlier step. Although you have a few options on how to get that data from the buffer and into a return value, the C function MemCpy works well for this. Given the address of the destination string, the address of the source data and it's length, it will copy that data into the string variable of your choice. In this case, even though the size of the value string is returned by VerQueryValue, MemCpy is instructed to only copy the first 128 bytes. The primary reason for this is that RVal is defined at 128 bytes max and the value strings are all null-terminated, so no harm is done even if the value is less than 128 bytes long.

The class and template:

The class files consist of "bdVer.inc" and "bdVer.clw". The API calls are defined in "bdVersion.lib", and the extension template is called "bdVerInfo.tpl".

The class files need to reside in the ABC LibSrc folder, usually C:\Clarion6\LibSrc or C:\C55\LibSrc.

bdVerInfo.tpl needs to reside in the Template folder, usually C:\Clarion6\Template or C:\C55\Template.

Finally, bdVersion.lib should reside in the Lib folder, usually C:\Clarion6\Lib or C:\C55\Lib.

It is important to note that this template works with either Clarion template or ABC template chain based applications. Remember, just because you're using the Clarion template chain for your applications, doesn't automatically mean you can't use class libraries and objects within your applications.

Register the template and you're ready to go. Choose a procedure that would benefit from this information like Main or About, and add the extension template to it. It's called "Recall Version Resource Info Extension Template".

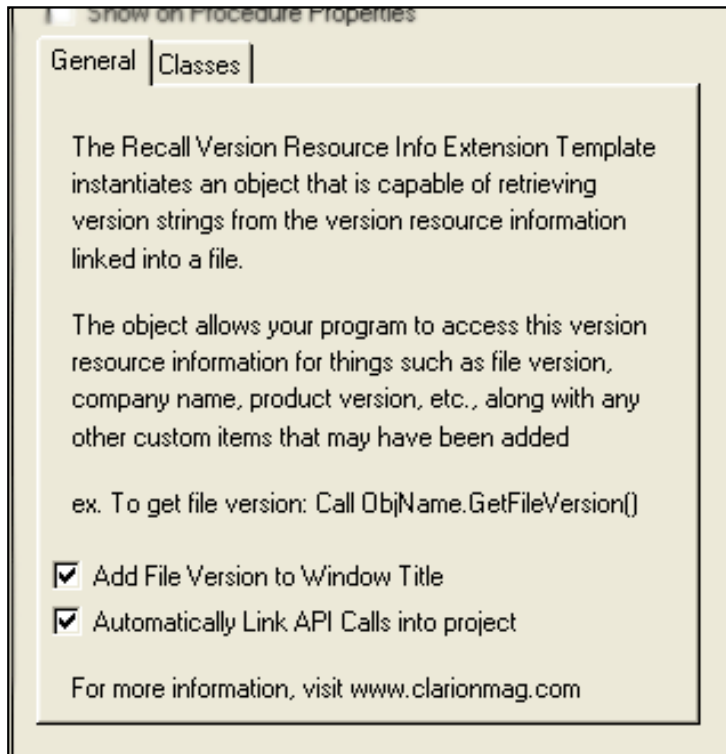


Figure 2. The template prompts

There are only two prompts on the template. If you want to have the template automatically add the file version to the window's title bar, check "Add File Version to Window Title".

If you are already using some sort of library that uses the three API calls mentioned before, you may get a link error talking about duplicate references. You can avoid this by unchecking the second box to prevent the template from adding "bdVersion.lib" to the project.

The template is responsible for including the class information in your program and creating an instance of the `bdVersionInfo` object. You can change the object name the template chooses by clicking the Classes tab and changing the entry by object name. You can also add new methods and derive many of the existing methods of the class.

Use of the object is quite simple. For example, if you want to retrieve the file version, a call to `ObjName.GetFileVersion()` returns a string containing that information. There is no need to call the `Init` method before using any of the other methods as it will be called automatically the first time any values are queried. All of the class methods that retrieve version resource information are listed below:

<code>GetComments</code>	<code>GetCompanyName</code>
<code>GetFileDescription</code>	<code>GetFileVersion</code>
<code>GetInternalName</code>	<code>GetLegalCopyright</code>
<code>GetLegalTrademarks</code>	<code>GetOriginalFilename</code>

GetProductName	GetProductVersion
GetPrivateBuild	GetSpecialBuild

All of the above methods return a `STRING`.

For user defined version strings, call the `GetInfo` method with the user defined string and it will return its value.

As I mentioned earlier, the class defaults to use the U.S. English language string and the multilingual code page, but you can override this behavior with a call to `SetLangID ()` and `SetCodePageID ()` if necessary. I've included equates for some of the common languages and code pages in the `bdVer.inc` file. For example to make the object look for German resource info, you would call `ObjName . SetLangID (bdLang : German)`.

There are a few other housekeeping methods as well pertaining to things such as retrieving the path and file name of the EXE/DLL, language and code page properties, and a variation of the `Init` method that allows you to specify a different path/filename instead of the currently executing file. You could use this to get information about specific DLLs, for instance, without having to declare the class in the DLL itself.

Take a look at the template file too. It demonstrates how to expose potential virtual methods within your classes to the embeditor for derivation, and allows the programmer to extend the class with new methods as well.

Gaining access to version resource information is now a point and click operation, thanks to a simple class library and template wrapper.

[Download the source](#)

Reader Comments

[Add a comment](#)

Clarion Magazine

Reborn Free

CLARION
online

Porting to the C6 Threading Model: The AutoLog Example

by Carl Barnes

Published 2004-01-16

Clarion C6 has gone gold and so it's time to start getting serious about porting applications. In this series of articles I will look at porting some of the C55 example applications to C6. This will provide some real world problems and real solutions, but on a small enough scale to not be overwhelming.

Threading issues

Prior to C6 Clarion used a cooperative threading model. The `ACCEPT` statement was used to ensure that only a single thread would be processing an event's code while all other threads waited for the `ACCEPT` to regain control. This made it very easy for a Clarion developer to write an app that had multiple threads. While this worked well most of the time, it did have enough limitations and problems that it had to go.

Clarion C6 implements threads the way 32-bit Windows intended, unlocking the RTL and allowing thread switches to occur preemptively. Your code can be interrupted at any time by a thread change, possibly leaving your data half baked and open to corruption. And remember that a single Clarion 4GL statement can encompass many lines of assembler code in the RTL. The thread change can occur anywhere in that assembler code. What appears to be a simple string assignment can require multiple four byte moves at the assembler level.

My primary focus here will be on threading-specific issues. The preemptive threading presents problems for unthreaded global data in use by multiple threads. There are two possible problems that can occur when your code does not protect itself from an untimely preemptive thread switch. The first is the code may not always function as intended, should another thread change a global variable. The second is the code may corrupt data when another thread only partially completes changing complex data values like strings, groups or queues.

Some operations, like integer and reference assignments are atomic, meaning they can't be interrupted by a thread change. But even then you have to be careful. What if an integer assignment is conditional based on some other data? An example of this is `IF Glo:X=1 THEN Glo:X=0:` Because a thread's code can be interrupted at any time, `Glo:X` may be changed to a new value

between the test `IF Glo:X=1` and the assignment `Glo:X=0`.

Because the thread change can occur at the assembly level there is a risk of data corruption. One thread could have partially changed a global string when a thread switch occurs and the new thread reads the partial data.

Actually it's more than simply global data. Any unthreaded static data could be a problem, and this encompasses Module data and local data with the `STATIC` attribute. So I'll refer to this global, module and static data as `GMS`. Clarion just makes Global data too easy, and the previous cooperative threading made it reliable and easy. It's a new world and more care is required in dealing with `GMS` data.

This is just a brief overview of the threading issues. For more details on the subject read the [Demystifying C6 Threading](#) series of articles on the SoftVelocity website. You may also want to download the Multi-Threaded Programming and Thread Model Changes PDFs from the SoftVelocity [web site](#). And there is a Thread Model FAQ section in the FAQ PDF with additional information in a Q&A format. Finally, the gold release CD install will include a previously unpublished "Migrating from Prior Versions" document that discusses the new threading model with and recommendations for porting.

Once you have a good (or even basic) understanding of the threading issues, you're ready to start converting your applications.

Port to C6 using C55

Many of the changes and techniques required for C6 work perfectly well in C55 (and prior versions). If this were production code I would attempt to make as many changes as possible and compile and test in C55. This way if I found a showstopper in C6 I would not lose all my changes reverting back to C55. (Since I am learning on examples in these articles I will work on them totally C6.)

For code that had to be different in C6 I would still try to implement it in C55 and wrap it in `OMIT/COMPILE` using the predefined compiler flags `_C60_` and `_C55_`. There is also a new flag in C6 named `_CWVER_` that returns the exact version number (6000 for C6 Gold). This is useful to library writers who must implement special code for specific releases.

How to Port to C6

To begin your port to C6 you first have to find all of the `GMS` data. You could hunt around in the IDE and find it. It's pretty easy to find the Global data under the Global button, only the `THREAD` attribute does not show in the list. But there could also be global data coming from the `DCT` or in a global embed. The best and most certain way to find `GMS` is to look at the generated source. You might say the buck stops there.

I suggest you create a text document that contains a list of all of the `GMS` data declared in the app. Do this by opening each source module and performing a copy/paste into this project data document. In the global module you will find all of your global data. At the top of each member module will be

any module data. You will also need to search your source for local data declared with the `STATIC` attribute.

One good way to do this is in the IDE Procedure Tree using the Module View. Right click on the Global Module (the first one in the tree) and select "Module" from the popup menu. This will open the global module in the Clarion editor. For each of the member modules select it and look at the right hand side (RHS). If you see data or embeds listed you will need to review that module's source. Right click on the module and select "Module" from the popup menu. You can find `STATIC` data by using the Search files menu item (a.k.a. TopToy) in the file menu of the IDE and searching `*.CLW` for "STATIC".

The primary data of interest in this article does not have the `THREAD` attribute. For a real app I would include `THREAD` data in my list and review it also with an eye towards eliminating it. Thread data is not implicitly good, it just has no problems with preemptive threading. Looking at the AutoLog example I found the below list of data without the `THREAD` attribute:

```
GLO:Report      STRING( 1 )
GLO:VIN         STRING( 20 )
GLO:StartDate  LONG
GLO:EndDate    LONG
```

Next, you have to know exactly how and where each variable is used so you can determine if it needs protection, and how you will provide that protection. And again the best way is to review all of the generated source code. You will need a text file search program that will let you find all usage and view the surrounding source quickly. While TopToy will work I, of course, use my own utility Clarion Source Search. Figures 1 and 2 show the results of a search for `GLO:Report` and a view of the source surrounding a line.

File	SubDirectory	Line	Location Text
autol001.clw		461	Main Menu::ReportMenu GLO:Report = 3
autol001.clw		466	Main Menu::ReportMenu GLO:Report = 4
autol001.clw		1679	SelectDateRange ThisWindow.TakeAccepted CASE GLO:Report
autol001.clw		1689	SelectDateRange ThisWindow.TakeAccepted GLO:Report = 0
autol001.clw		2389	BrowseEXP:AutoVINKey ThisWindow.TakeAccepted GLO:Report = 2
autol002.clw		416	BrowseTRI:AutoVINKey ThisWindow.TakeAccepted GLO:Report = 1
autolog.clw		29	GLO:Report STRING(1)

Results Files 7 found in 0.22 seconds, 254,830 bytes

Figure 1. Files with GLO:Report

```

View autol001.clw - line 461 - (1 of 7)
Main Menu::ReportMenu
OF ?PrintSER:ServDescKey
  START(PrintServiceListing, 050000)
OF ?PrintAUT:AutoClubNameKey
  START(PrintAutoClubs, 050000)
OF ?PrintINS:CompanyKey
  GLO:Report = 3
  START(PrintInsCo, 050000)
OF ?ReportsPrintExpenses
  SelectDateRange
OF ?ReportsPrintTrips
  GLO:Report = 4

```

Figure 2. Viewing the source code

A review of the source for each of the four variables found the following:

1. GLO:Report is used to pass the report selected to a newly started thread
2. GLO:VIN is used to pass the Vehicle Identification Number of the car selected in a browse to a report in a new thread
3. GLO:StartDate and GLO:EndDate are used to pass the dates entered between procedures on the same thread (see Figure 3).

Passing Data to New Threads

The source shows that the variable GLO:Report is used to pass the user's report selection to a newly started thread. Passing data to new threads is a very common use of global and module data with Clarion. Store a value in a global variable, start the thread and the new thread saves the global value into a local or thread variable.

In reality as long as the thread is started by a menu selection, it is not very likely that the user could start two threads so quickly that there would be a problem in C6, especially if the values are set just before the START and saved immediately by the new thread.

That said, doing so leaves a window for possible problems, and it's a poor foundation to build on so I don't recommend "hoping" nothing goes wrong. It could happen that six months down the line the customer asks you to make it possible for an overnight process to run a bunch of reports. In this code you start up six reports at once and that could open the window to problems. A solid foundation is worth some extra time.

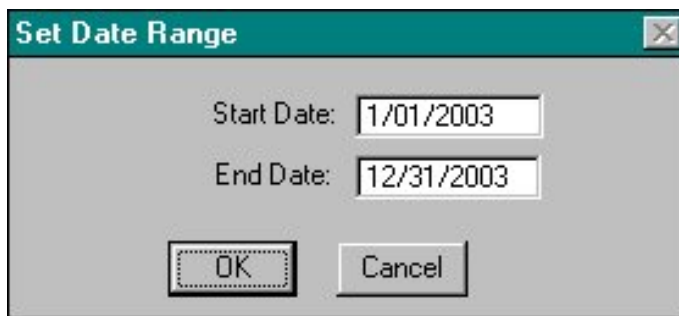


Figure 3. Setting the date range.

In AutoLog the variable `Glo:Report` is assigned in four places, so later on the procedure `SelectDateRange` knows what report procedure to run after the user enters his dates. Here's the code in `SelectDateRange` that calls the report procedure when the Ok button is pressed:

```

OF ?OkButton
  CASE Glo:Report
  OF 1 ; VehicleTripRpt
  OF 2 ; VehicleExpRpt
  OF 3 ; PrintExpenses
  OF 4 ; PrintTrips
  END
  Glo:Report = 0

```

On examining the code I think that `Glo:Report` has been weakly implemented, and would have issues even in C55. I'll explain why in a moment. As the above code shows the global variable is used to call the report after the user presses the OK button. The value of `Glo:Report` should have been saved to a local variable. If a second report is started before the date selection is complete the value of `Glo:Report` will be changed and the user will get two copies of the last report selected.

The best move for data in this category is to get rid of the globals and pass the data to the procedure. This reduces memory use and prevents corruption. `SelectDateRange` could be prototyped (`STRING Glo:Report`) and the desired report number passed in the `START` command as a parameter. In reviewing further I found that `Glo:VIN` would also have to be handled this same way for two reports.

The way AutoLog runs reports using `SelectDateRange` as a middle man is, I think, the wrong way to do it. And passing other parameters, like `Glo:VIN`, through the middle man will just grow in complexity over time. The right way to do this is to start the report procedure and have the report call `SelectDateRange`. The changes I will make to Autolog are:

1. Delete the `Glo:Report` global declaration
2. There are four places where `Glo:Report` is assigned a report number and `SelectDateRange` is called or started. I will change them all to start a new thread. The code looks like this:

```

OF ?ReportsPrintTrips

```



```
GLO:Report = 4  
SelectDateRange
```

I will delete the GLO:Report line will and START the desired report procedure.

```
OF ?ReportsPrintTrips  
START(PrintTrips,25000)
```

Main|Print All Expenses will start PrintExpenses

Main|Print All Trips will start PrintTrips

The BrowseEXP:AutoVINKey print button will start VehicleExpRpt

The BrowseTRI:AutoVINKey print button will start VehicleTripRpt

3. The four report procedures need to call SelectDateRange using the below code in the each report's ThisWindow.Init "Initialize the procedure" embed point:

```
SelectDateRange  
IF GlobalResponse<>RequestCompleted  
RETURN Level:Notify  
END
```

4. The SelectDateRange OK button no longer calls the reports so they need to be removed from the Procedures list. When OK is pressed it needs the following embed code to return a response to the call added above in step 3.

```
ThisWindow.Response = RequestCompleted  
POST(EVENT:CloseWindow)
```

The variable GLO:VIN also fits into the category of a global passing data to a new thread. It is set to the VIN of the currently selected record in a browse and a report is started on a new thread. Even in C55 this code could have trouble if a second report was STARTed for a different VIN. The best bet is to change the code to pass the VIN to the new thread as a parameter and get rid of the global.

GLO:VIN is used to limit the range of the report by specifying a range type of Single Value under the Report Properties Range Limits. The Single Value method requires that the Range Limit Value variable be defined in the IDE so to continue using this method would require defining a VIN variable locally and assigning it the passed VIN. Rather than do that the simpler way, implemented below, is to change to the Current Value method and set the current value to the passed value.

1. Delete global GLO:VIN
2. Change VehicleExpRpt
 - a. Prototype/Paramaters to (STRING VIN2Report)
 - b. Report Properties range type to "Single Value"

- c. Assign the current value with the code `EXP:AutoVIN=VIN2Report` after the Open Files embed
3. Change `BrowseEXP:AutoVINKey` to pass the VIN by putting `(EXP:AutoVIN)` into the parameters entry for the print button actions that start `VehicleExpRpt`.
4. Change `VehicleTripRpt`
 - a. Set Prototype/Parameters to `(STRING VIN2Report)`
 - b. Set the Report Properties range type to "Single Value"
 - c. Assign the current value with the code `TRI:AutoVIN=VIN2Report` after the Open Files embed
5. Change `BrowseTRI:AutoVINKey` to pass the VIN by putting `(TRI:AutoVIN)` into the parameters entry for the print button actions that start `VehicleTripRpt`.

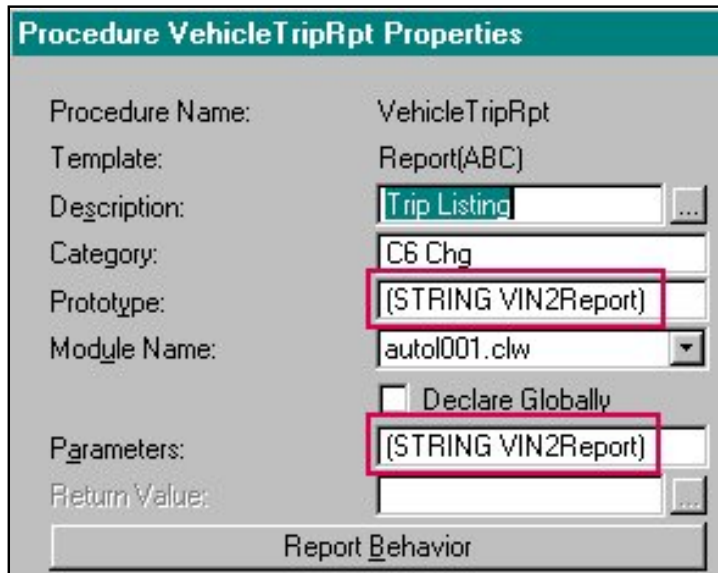


Figure 4. Changing the prototype

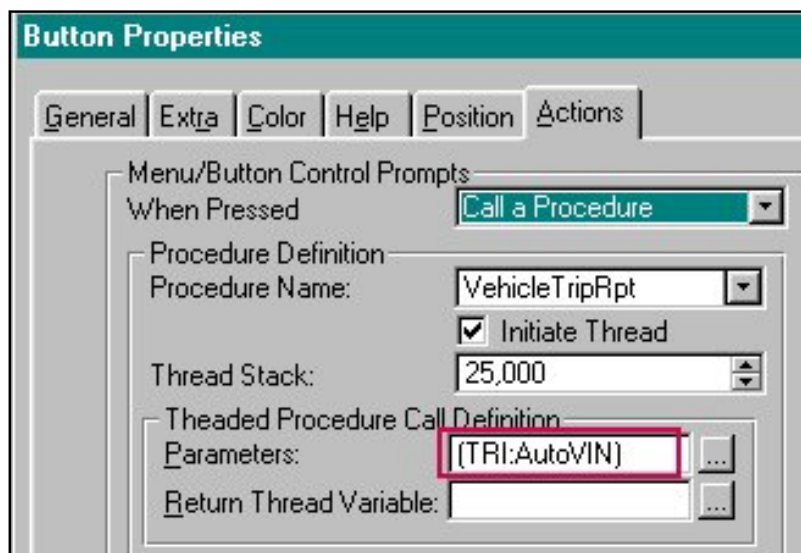


Figure 5. Passing a parameter to the threaded procedure

That's all for this week. [Next week](#) I'll discuss sharing data between procedures.

[Download the source](#)

[Carl Barnes](#) is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

[Add a comment](#)

Excellent article, Carl! I am biased towards articles...

Great article. I'm looking forward to more. One that...

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



Porting to the C6 Threading Model: The AutoLog Example Part 2

by **Carl Barnes**

Published 2004-01-22

[Last week](#) I began by discussing some of the basic issues of porting applications to the C6 threading model, and began converting the AutoLog example app. I also showed how to pass data directly to STARTed procedures instead of using global data.

Another common use for global data is to share data between procedures in a calling chain. In the AutoLog example `GLO:StartDate` and `GLO:EndDate` are used to enter dates in the `SelectDateRange` procedure, and to filter records in four report procedures. If all of the procedures are on the same thread, the simplest fix is to add the `THREAD` attribute to the date. For the dates in AutoLog this will work perfectly. Actually this should have been done in the C55 version or a bug would result if the user STARTed a second report and entered different dates. After this change the generated date looks like this:

```
GLO:StartDate LONG,THREAD
GLO:EndDate LONG,THREAD
```

A better method would be to change the `SelectDateRange` prototype to (`*LONG StartDate, *LONG EndDate`) so the range values can be returned to the caller. Then the global variables can be eliminated. This results in less thread data and a more solid design. But in the real world I would probably not risk breaking working code by this very small enhancement.

One enhancement I would make is for the date selection to default to the previous dates I entered in this run session. Currently it always defaults to the maximum date range. To do this I would declare two `STATIC` local variables in the `SelectDateRange` data. Why not declare them as globals? Because currently they are only needed in a single procedure, so the best choice is to limit their scope and declare them locally. By doing so I also avoid a global recompile. Below you see them defined with the desired default initial values:

```
Default:StartDate LONG(36528),STATIC !01/01/1901
Default:EndDate LONG(109211),STATIC !12/31/2099
```

The procedure initialization code needs to change to assign the initial values for the date fields from the new static default values using this code:

```
GLO:StartDate = Default:StartDate
GLO:EndDate = Default:EndDate
```

When the user presses the Ok button the values he entered must be stored into the new default variables using this simple code:

```
Default:StartDate = GLO:StartDate
Default:EndDate = GLO:EndDate
```

But wait! Since these default variables are `STATIC` without the `THREAD` attribute could an untimely thread change cause corruption? The short answer is no. For the simple data types `BYTE`, `SHORT`, `USHORT`, `LONG` and `ULONG` the data is `DWORD` aligned and the assignment happens in a single operation, so data corruption cannot occur. But there is one caveat. If these data types are contained in a structure (`Group`, `Record` or `Queue`) then the structure is `DWORD` aligned and the data inside is packed together. This means that the individual fields in the group may not be `DWORD` aligned and the assignment will not happen in one operation.

The other problem that could occur is a thread switch between setting the start date and setting the end date. Ideally these should be done together in a matched pair. Given the way this code works in response to user actions it seems extremely unlikely a problem could occur. But this article is about doing it right and the best way to protect this data would be to wrap the sections of code in a `CriticalSection`. A `ReaderWriterLock` could also be used, but would be overkill and require more code.

To use any of the new thread synch library functions you must have an `INCLUDE('CWSYNCHM.INC')`, `ONCE` in your global module. Then in the global data embed the following code defines a `CriticalSection` interface to protect the default data:

```
DefaultCS &ICriticalSection
```

In the `Main` procedure at startup the interface is created using this code:

```
DefaultCS &= NewCriticalSection()
ASSERT(~DefaultCS &= Null)
```

Back in `SelectDateRange` the setting of the current values from the default values is protected by wrapping it in a critical section:

```
DefaultCS.Wait
GLO:StartDate = Default:StartDate
GLO:EndDate = Default:EndDate
DefaultCS.Release
```

And likewise the saving of the new entries as default values is protected:

```
DefaultCS.Wait
Default:StartDate = GLO:StartDate
Default:EndDate = GLO:EndDate
DefaultCS.Release
```

A slightly different way to implement this would be to declare the `ICriticalSection` in `SelectDateRange`. I prefer this approach which keeps the data and the synch objects to protect the data in the same scope. This critical section must be defined as `STATIC` and `NEWed` only once and not disposed. A further complication is that two threads could try to `NEW` the critical section at the same time so that section of code should be protected by a globally declared critical section. That's a story for another time....

C6 Adds BINDEXPRESSION

What fun would this be if I didn't look at some C6 enhancements? One interesting addition to the Clarion language is the new `BINDEXPRESSION()` statement. The pre-C6 statement `BIND('GLO:StartDate', GLO:StartDate)`

causes the RTL evaluator to lookup the *current* value of `GLO:StartDate` every time the filter is evaluated. It is binding the `Address(GLO:StartDate)` to the literal `'GLO:StartDate'`. But the values of the start and end dates do not change during the report, so there is some unnecessary code running to lookup the current value of these dates for every record when the filter is evaluated.

In C6 you can change this code to `BINDEXPRESSION('GLO:StartDate', GLO:StartDate)` – this way the literal `'GLO:StartDate'` is bound to the value of the expression `GLO:StartDate`, i.e. the actual date serial number. This is a "squeezing the last drop" type of thing and so you will probably see no significant performance difference. But it remains a better way to handle a constant value. In the attached AutoLog app you can see this implemented in the `PrintTrips` procedure. I removed the dates from the Hot Fields tab and added `BindExpression` using another new C6 feature, the "Bind Fields and Procedures" button on the Procedure Properties window.

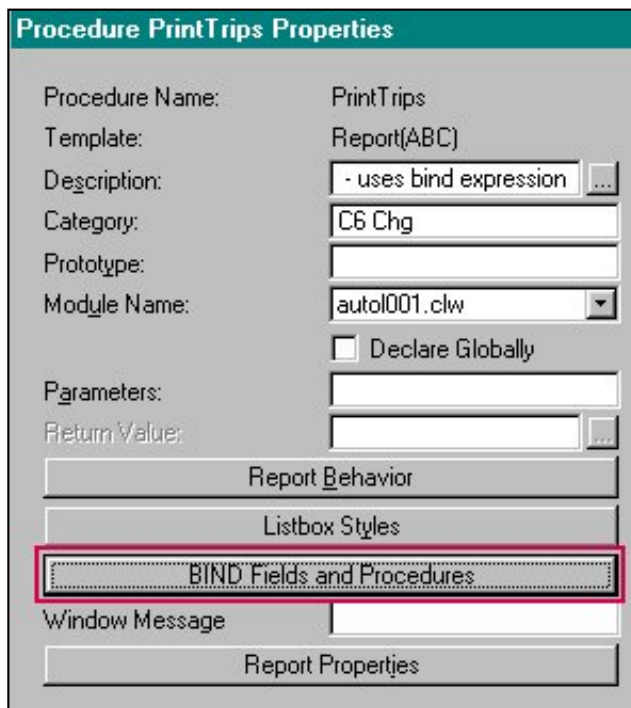


Figure 6. The BIND Fields and Procedures button

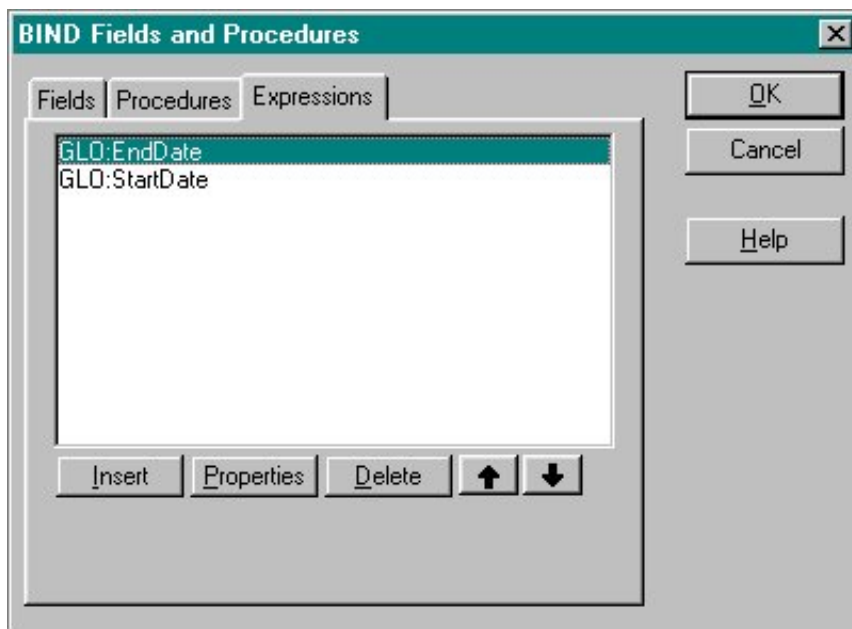


Figure 7. The BIND Expressions list.

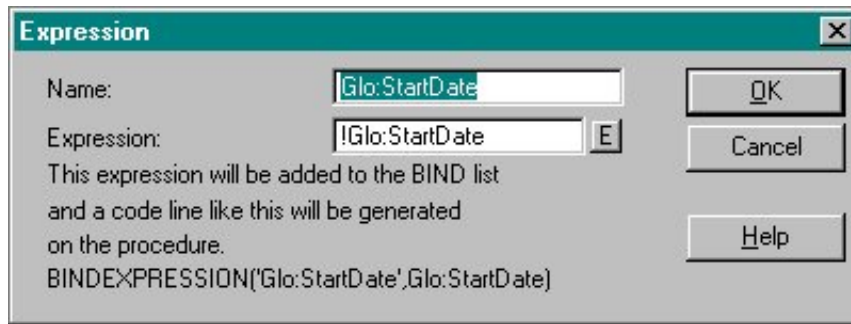


Figure 8. Creating a BINDEXPRESSION.

For this to work, the call to `SelectDateRange` that enters the report values must be performed before the Bind Variables embed in `ThisWindow.Init`. In the above changes I placed the call to `SelectDateRange` in the "Initialize the procedure" embed point. This occurs after the template generated BIND statements and so the generated BINDEXPRESSION would not have the entered date values. So the call to `SelectDateRange` needed to be moved up a bit in `ThisWindow.Init` before the "BIND variables" embed point at priority 5101.

In trying to move the call to `SelectDateRange` to an embed before BINDEXPRESSION I ran into a "feature" of C6. Even though I deleted the dates from the Report Properties Hot Field tab the templates continued to generate a BIND for them. What I finally figured out is that C6 has a new "Auto Bind" feature that parsed the filter expression (`TRI:BeginDate >= GLO:StartDate` and `TRI:BeginDate <= GLO:EndDate`) and automatically ensured there was a BIND for all fields in the filter. The new Bind Fields button has a tab named Template Defined that lists the BINDs added automatically. It allows checking a box for each field you do not want to auto BIND. This checkbox did not work for me and would frequently get cleared by the templates during generation.

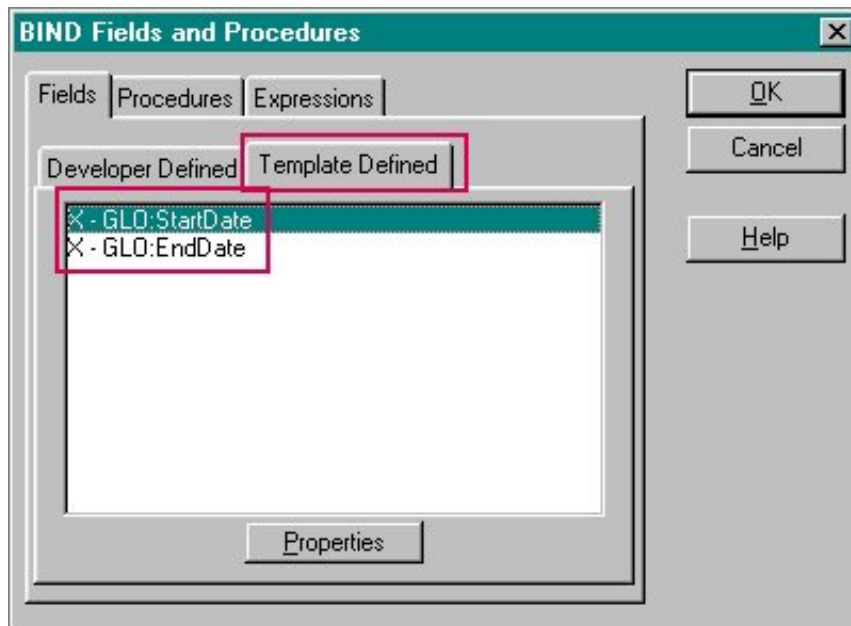


Figure 9. The Template Defined tab

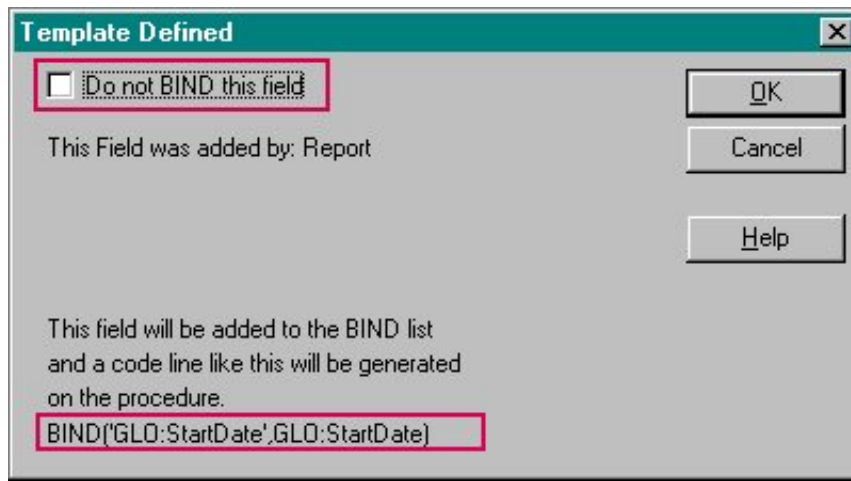


Figure 10. Bind settings

I gave up trying to fix this and reported the bug to SoftVelocity. Since my BINDEXPRESSSION comes after the BIND in the generated code it should replace the BIND and work as I desired. The incorrect generated code for PrintTrips looks like this:

```
! BIND variables
BIND( 'GLO:StartDate',GLO:StartDate)
BIND( 'GLO:EndDate',GLO:EndDate)
BINDEXPRESSSION( 'GLO:StartDate',GLO:StartDate)
BINDEXPRESSSION( 'GLO:EndDate',GLO:EndDate)
```

While the above code should work I thought it was a bad idea to leave something potentially confusing. Some times the best way to solve a problem is to walk away from it for a while. Ten minutes outside adjusting the sump pump hose and the light bulb turned on. The solution is to not *name* the expressions the same as the variables. Changing the BINDEXPRESSSION names to simply StartDate and EndDate (and the filter too) removed the auto-generation of the undesired BIND statements. The generated lines are:

```
BINDEXPRESSSION( 'StartDate',GLO:StartDate)
BINDEXPRESSSION( 'EndDate',GLO:EndDate)
ThisReport.SetFilter('TRI:BeginDate >= StartDate and TRI:BeginDate <= EndDate')
```

Here are a few things I learned in my experimenting that are not obvious from the documentation. BINDEXPRESSSION has two forms: Constant and Expression. Above, I use the former and bind a name to a constant date value. I did this with the templates by prefixing the expression with an exclamation point. I would do it in hand code by *not* name wrapping the second parameter in quotes, e.g. BIND('GreekPI', 3.14159).

Wrapping the second parameter in quotes, e.g. BINDEXPRESSSION(' Total', 'Quantity * Price') makes it an expression that must be run through the Evaluate() engine. That means that all the variables used in the expression must be bound or BINDEXPRESSSION will return a bind error. The bind errors have changed from 800-801 to 1010-1012 in C6. And the C6 error message text for these errors will tell you exactly what variable names could not be found. All in all, you will probably spend less time in C6 chasing a forgotten BIND of a variable. But the current code generated by the new procedure bind templates does not trap errors for BINDEXPRESSSION or auto-bind the variables used in the expression.

Summary

This wraps up converting AutoLog for C6 preemptive threading. The converted version is available for download below, and was created using the gold patch.

Key points to remember:

1. Review your source declarations and build a document containing a list of all of your Global, Module and Static data.
2. Review all of the source code that uses the variables to understand how and where they are used.
3. Try to get rid of non-THREAD data when possible. Try to get rid of THREAD data too.
4. If the data is being used to pass values to a new thread try to pass the values as parameters
5. Try to make the data work by adding the THREAD attribute.

[Download the source](#)

[Carl Barnes](#) is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

[Add a comment](#)

**Hi Carl, Unless I'm missing something, the simple...
I was talking about the potential for "corruption" of the...**

Clarion Magazine



Clarion News

[Search the news archive](#)

RPM6 for Legacy(Clarion) Available Now

The legacy add-on for RPM6(ABC) is now available. This release provides support for those developers maintaining legacy applications in C6. Also, an updated RPM6(ABC) install should be available within the next few days. This update includes a few minor fixes to the ABC templates.

Posted Wednesday, January 28, 2004

FUSE Products C6 Compatible

All of ThinkData's FUSE products are now Clarion 5.5 and Clarion 6 compatible. The FUSE products are Component Object Model (COM) wrappers built to provide native COM automation to popular business applications. ThinkData's current product line offers integration with MS Outlook, Quickbooks & XML. Products currently under development include: actFUSE, Clarion COM Automation for ACT! and FaxFUSE, Clarion COM Automation for WinFax/Microsoft Fax/RightFax

Posted Wednesday, January 28, 2004

OutlookFUSE 2.0

ThinkData has released OutlookFUSE 2.0, a native COM (Component Object Model) automation interface to Microsoft Outlook using Clarion 5.5 and Clarion 6. The new version 2.0 includes the following new features: Full compatibility with Outlook 2003, including the interception of Outlook events by your Clarion application; Updated installer which allows the installation of xmlFUSE on machines using both C5.5 and C6.

Posted Wednesday, January 28, 2004

XMLFUSE 2.0

ThinkData has released xmlFUSE 2.0, a native COM (Component Object Model) automation interface to the Microsoft XML (Extensible Markup Language) 4.0 SDK and the Microsoft SOAP (Simple Object Access Protocol) 3.0 SDK using Clarion 5.5 and Clarion 6.

Posted Wednesday, January 28, 2004

ThinkData 25% Off Sale Through Feb 1

For the next 7 days (January 26 - February 1), ThinkData is offering 25% off each of its products. This would result in a savings of \$179.00 if you purchase OutlookFUSE, xmlFUSE,

qbFUSE and EScroll.

Posted Wednesday, January 28, 2004

[EasyAutoEntry 1.03](#)

EasyAutoEntry ver 1.03 is now available. Changes include: In the "Suggested Part" style you may select a variant and delete it from the list with the Alt+Delete key; Fixed global tab drop setting; Fixed loss of focus problem. This is a free upgrade for all registered users.

Posted Wednesday, January 28, 2004

[xAppWallpaper v1.5](#)

xAppWallpaper v1.5 is now available. New in this version: Six button control templates for xAppWallpaper features were added. Updated demo.

Posted Wednesday, January 28, 2004

[Clarion Training Bookings In Centurion, Pretoria](#)

Clarion training bookings are now open for training located in Centurion, Pretoria. This class will be using a PC lab so there will be no need to bring our own PC or Notebook. Courses include Moving to Clarion 6, Essentials Clarion 6, and MS SQL and Clarion 6. To book your seat(s), please email raymond@soggy.co.za.

Posted Wednesday, January 28, 2004

[BoxSoft Super Stuff 6.10](#)

BoxSoft Super Stuff 6.10 is now available. Changes include: The Clarion 6 compatible version of the "MikeHanson" template chain now included - this was formerly the public domain template; The legacy mhView templates have been enhanced, including support for CPCS; Quicken-style date entry feature in the legacy chain; The legacy resize templates are more stable than the old PD version; Changed "Add Max Button" setting to display warning if the developer didn't manually add the MAX attribute to the window format; Added support for MultiProject. This is a free update for developers who have already upgraded to Super Stuff 6.0. The password is the same. For all purchase and upgrade issues (including new passwords), please Mitten Software. Their e-mail address is Mitten@MittenSoftware.com, and their phone numbers are (800) 825-5461 and (952) 745-4941.

Posted Wednesday, January 28, 2004

[ClarionPost.com Temporarily Unavailable](#)

ClarionPost.com will be unavailable for several days due to a hard disk crash.

Posted Wednesday, January 28, 2004

[Ready-to-go New Products](#)

1stLogoDesign's list of Ready-to-go products is growing. There are new additions to logos, web templates, splash screens, images, and buttons. Customization available.

Posted Wednesday, January 28, 2004

[SelfService v1.11](#)

CapeSoft SelfService v1.11 beta is now available. SelfService allows your Clarion 6 applications to be run as Services in Windows NT, 2000, XP and 2003. Close to gold release.

Posted Wednesday, January 28, 2004

[ZipApp 1.2b](#)

This minor revision includes a requested change for HLP option to include HM* files.

Posted Wednesday, January 28, 2004

[Richard Rose Heads Clarion User Group](#)

The UK Clarion User Group is now up and running and is under the guidance of Richard Rose. The user group is looking for members and wants their input into what the User Group should be about, i.e. training, people networking, product demos, special offers etc. To join up and/or see the schedule for the next meeting on March 8th visit the web site.

Posted Wednesday, January 28, 2004

[Web Site Reviews](#)

With the recent interest (not to say animated discussion) in the SV newsgroups re web site ratings, Arnor Baldvinsson has set up a forum on the Icetips bulletin board where members of the Clarion community can ask for reviews/critique of their web sites.

Posted Wednesday, January 28, 2004

[Redirection Test Utility Program](#)

The REDRTEST.EXE program is a diagnostic tool that can be used to test for current redirector software on network workstations and most file servers. It also tests for our recommended Windows registry settings including Windows NT/2000/XP opportunistic locking settings and can be used to implement those registry settings if they need to be changed. This program will also optionally implement recommended settings for workstations using the Novell Client 32 Redirector.

Posted Wednesday, January 28, 2004

[Taboga Software Holiday Schedule](#)

Edgard L. Riba is on holiday from Jan 17 - Jan 30, and email answering will be on very low priority during those dates.

Posted Tuesday, January 20, 2004

[Training for Gauteng, SA](#)

The training schedule for Gauteng, SA, has now been posted, and bookings are available.

Posted Tuesday, January 20, 2004

[Fenix Customer Quotes](#)

This page contains quotes from customers using the Fenix ASP.NET Generator.

Posted Tuesday, January 20, 2004

[EasyAutoEntry 1.02](#)

EasyAutoEntry 1.02 is now available. Changes include: Added ability to filter suggested variants; Added filter template; Fixed bug with SDI windows which caused the EXE to remain in the task list after exit. This is a free upgrade for all registered users. The demo version includes the demo template and the demo library. It will allow you to test all features of EasyAutoEntry, but will automatically add variants of the string "demo" to the text.

Posted Tuesday, January 20, 2004

[ClarionNET Developers Demo System](#)

The complete ClarionNET 1.4 development system is now available for download as a demo. This system is exactly what you would purchase but without the unlock codes. The only restriction is that a "Unlicensed Demo Version" message will pop-up when you connect from the remote client.

Posted Tuesday, January 20, 2004

[BST,BoTpl C6 Patch](#)

A change in the system variables in C6 has caused all the BigTamer templates (using a common group) to sometimes incorrectly identify the template chain for C6 when using the Clarion Template Chain. This includes the free BoTpl templates. If you Use the C6 Clarion Template Chain you need this patch. 196k installer download.

Posted Tuesday, January 20, 2004

[Free Point Of Sale Software Written In Clarion](#)

Patrick De Laet has released a free POS system with an Office 2003 look and feel. TT-POS uses MSDE as its database system. The web site and software are both in Dutch, although the software is fully translatable.

Posted Tuesday, January 20, 2004

[ClarionNET 1.4 Release](#)

ClarionNET 1.4 and Server Deployment Manager 1.4 are now available for download. All users are requested to register at the ClarionNET web site, which is the new home of ClarionNET. Version 1.4 is a free maintenance release. Nothing really major is included except 128 bit encryption and minor internal improvements. The internal version number is still "78" because there are no incompatibilities with ClarionNET 1.3 that require a recompile of the client program. This release is only for Clarion 5.5. Support for Clarion 5.0 has been dropped. The Deployment Manager version 1.4 has received extensive work in the area of server polling and application broker launching. In addition the ClarionNET server DLL (distributed with ClarionNET) has received some internal changes that result in far less disk activity when the system records performance information. Also new is the ability to specify an external IP address for the application server so firewalls can do port forwarding.

Posted Tuesday, January 20, 2004

[Clarion Jobs Page Updated](#)

Alfred Blaho has listed some new Clarion jobs listed on this page.

Posted Tuesday, January 20, 2004

[Firebird ODBC Beta 2](#)

Kelvin Chua reports that the Firebird ODBC Beta 2 is now available. The known variant 'fn' had been added.

Posted Tuesday, January 20, 2004

[Ready-to-go Web Templates](#)

1stLogoDesign has released its first set of ready-to-go web templates. Just add your text and

whatever else you want. This is the first set of what will grow to be a large catalogue of web templates.

Posted Tuesday, January 13, 2004

[Big gCal Sale](#)

For a limited time save \$50 on the DLL version of gCal, now just \$49, and save \$100 on the source code version, now just \$199. gCal includes nine calendars, a date calculator, a control calendar, which you can use directly in your procedure, without pop-ups. gCal also includes a time picker, 35 date/time functions and a utility to customize your calendars.

Posted Tuesday, January 13, 2004

[In-Memory Database Driver Information](#)

Information on SoftVelocity's upcoming In-Memory Database Driver is now available. This is a new file driver technology that does not use physical tables for working with data. All data is stored in Random Access Memory (RAM), which gives the driver a number of unique properties.

Posted Tuesday, January 13, 2004

[CHT Website Redesign Completed](#)

The Clarion Handy Tools web site has a new, cleaner look.

Posted Tuesday, January 13, 2004

[Soggy Systems Products](#)

Soggy Systems is releasing the tools that have been used in their in-house projects as templates. These templates will work with Clarion 5.5 and 6, support ABC only, and have no DLLs. Free updates and upgrades. Templates include: Soggy Queues; Soggy Fields; Soggy XP Buttons; Soggy Flash; Soggy Settings; Soggy Bundle (Queues + Fields + Buttons + Flash + Settings); Soggy RRM. Documentation is being finalized, and the products will be ready to ship on the 12th of January 2004.

Posted Tuesday, January 13, 2004

[BST Ver 2.6](#)

Free upgrade for Version 2.5. Includes two bug fixes, three new templates, Dynamic Wall Calendar, refresh button, and small static wall calendar, plus 12 new features added involving synchronization, weeks starting on Monday, and adding your own items to the popup menus.

Posted Tuesday, January 13, 2004

[Free Popup Alarm](#)

John Griffiths points out this handy little popup alarm program written in Clarion that you can have at no charge, written by a programmer in Perth.

Posted Tuesday, January 13, 2004

[BoxSoft Super Invoice 6.04 Released](#)

Super Invoice 6.04 is now available, and includes the following changes: When using an existing CHECK control for EIP, it wasn't aligning with the in-list display icon; (ABC) "Count" Total type created empty LIKE() in QUEUE; (Clarion) "Total On" source variables sometimes were not projected in the VIEW and QUEUE; (Clarion) Typo in AcceptAll

variable. This is a free update for developers who have already upgraded to Super Import-Export 6.0 The password is the same. You can download the new file from www.boxsoft.net. For all purchase and upgrade issues (including new passwords), please contact Mitten Software. Their e-mail address is Mitten@MittenSoftware.com, and their phone numbers are (800) 825-5461 and (952) 745-4941.

Posted Tuesday, January 13, 2004

[BoxSoft Super Import-Export 6.03 Released](#)

Super Import-Export 6.03 is now available, and includes the following change: Fixed problem with DEFORMAT and large strings, which could cause numeric values to be imported as zero. This is a free update for developers who have already upgraded to Super Import-Export 6.0 The password is the same. You can download the new file from www.boxsoft.net. For all purchase and upgrade issues (including new passwords), please contact Mitten Software. Their e-mail address is Mitten@MittenSoftware.com, and their phone numbers are (800) 825-5461 and (952) 745-4941.

Posted Tuesday, January 13, 2004

[ADDA 1.0.0 Released](#)

ADDA (Advanced Data Dictionary Architect) 1.0.0 is now available. Changes include: Connection Data Link dialog is now defaulted to MS SQL Server; Connection information is kept between imports; As a freebie for .NET developers, added documentation for utilities contained in Afiq.Global assembly: Afiq.Global.chm. ADDA is a multipurpose toolkit for designing, creating and maintaining the database layer throughout the entire application lifecycle.

Posted Tuesday, January 13, 2004

[Clarion Resources And News Page](#)

Ingasoftplus has a new Clarion Resources and News page. You can use Search, Calendar, Categories, Month Archives etc.

Posted Tuesday, January 13, 2004

[News From 1st Logo Design](#)

There are a number of changes at 1st Logo Design: 1) Pricing on the logo packages is up, but you also get more. In addition to display logo formats you now also get ready to print format including grayscale. 2) New promotion. Every logo package (including the developers special) receives a free hat. Show off your new logo. 3) New developer's promotion. CD label design and jewel case insert or DVD insert design are now included. There are many enhancements and upgrades that you can add to the package. 4) All web design packages now include free submission to over 300 search engines and directories. 5) All web design packages now include free domain registration for 1 year. 6) New Ready-to-go logos. A new set of ready to go logos is now available and they are still \$99 each. 7) New Ready-to-go web templates. The first set of ready to go web templates is now available. We will be adding new templates during the next few weeks. Get them as low as \$59, includes html and psd files.

Posted Tuesday, January 13, 2004

[Clarion Training In Capetown](#)

There four seats open for Moving to Clarion 6 and one for the MS SQL course. Bookings will

closed on 15 January 2004.

Posted Tuesday, January 13, 2004

[Vote For etc 2004 Sessions/Classes](#)

Lee White has set up a poll for potential session and class topics for etc 2004. If you are seriously considering attending etc in June, please take a few minutes to let Lee know what topics you prefer.

Posted Thursday, January 08, 2004

[ClarioNET User Registration](#)

All ClarioNET users who purchased the product from SoftVelocity are requested to register themselves at the ClarioNET site. Entry of the "License Password" in the form is essential to verify that you have a license. A free Version 1.4 ClarioNET update will be available shortly and Michael Brooks needs to know who you are!

Posted Tuesday, January 06, 2004

[1st Logo Design Price Increase](#)

Effective January 1st the pricing for the logo packages has been increased.

Posted Tuesday, January 06, 2004

[Clarion Developers Challenge Grand Prize Winners Announced](#)

In a long season, spanning over 17 weeks, the overall winner of the Clarion Developers Challenge came down to the final game of the final week. This year's winner and NEW reigning Champion of the Clarion Developers Challenge Football Contest goes to Jerry Davis. Jerry finished this year with a total score of 140. Dean Burgess finished second with 138. And with a strong finish, Jason Johnson almost overtook the leaders and finished a close third with a score of 137. In the overall Encourager Award competition, Bob Naffziger won with a total score of 50. The Encourager Award is awarded to the person that competed each week from Week 10 through Week 17 and finished with the lowest combined score from those weeks.

Posted Tuesday, January 06, 2004

[Clarion Developers Challenge Week 17 Winners Announced](#)

First place for Week 17 of the Clarion Developers Challenge goes to Eddie Sizemore. Eddie picked a solid 11 out of 16 to claim first place in this week's contest! Tom Ferguson, Jason Johnson, Sebastian Streiger, Ron Childs, Dave Beggs, Jeff Nordgren and Shane Vincent all finished a close second with 10 correct picks. For finishing first in this week's Clarion Developers Challenge, Eddie will be receiving a copy of gQ donated by Jesus Moreno and Gitano Software. Eddie will also be receiving a copy of Product Scope 32 PRO, single user, spreadsheet license from David Troxell and Encourager Software.

Posted Tuesday, January 06, 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.

Reborn Free

CLARION
online

Addressing Arrays On A Window

by **Steven Parker**

Published 2004-01-23

Kelly Major, Fred Shepherd, Randy Goodhew, Jeff Slarve and Maarten Veenstra, thank you. Those kind Clarioneers walked and talked me through the problems I am about to discuss. They held my hand until it was solved.

The Problem

Without a doubt, the biggest part of the problem is that I loath, hate and despise arrays. I am convinced that arrays were created by people who would grow up to be hackers (in the bad sense) and writers of viruses.

Perhaps my attitude is explained by the fact that arrays had not yet been "invented" when I took my Computer Science course. To get the effect of what is now called an array, we had to use nested loops. You can, of course, imagine the ... uh, interesting results I often encountered. Perhaps this is why I understand that arrays are evil.

Nope, I just don't like arrays. I don't like them at all.

But, sometimes I have to use them, like it or not. Sometimes they are the best way to accomplish a goal. Payroll Deductions, in my current project, is a case in point.

My company sells its POS product to a lot of hospital gift shops. These gift shops allow employees to charge purchases. Payment is made by payroll deduction. To implement this, my software periodically loops through the employees' balances and exports a file of payments for the hospital's payroll system.

Some hospitals allow employees to "split" their payments. That is, full payment is not always required; payments can be spread out over time. Two methods of calculating the payment are available: Variable (which most hospitals use) and Fixed.

In a Variable payment scenario, the balance is divided by a factor. Figure 1 shows the setup window for this.

Payroll Deduction Split Setup

Automatically Split Payroll Deductions:

All (Everyone pays split payroll deduction)
 None (Everyone pays their full balance)
 Some (Each person is set up individually)

When "Some" is selected, you have the option to globally change all customers to either "Yes" or "No." You can then selectively change individual customers using the customer maintenance window.

Set all customer to Yes Set all customers to No

Payment Type:

Fixed Payment is fixed, as defined by their balance.
 Variable Payment is calculated by dividing their balance by the number of splits that are defined.

	Current Balance		Payment
Split 1:	0.00	- 0.00	FULL
Split 2:	0.00	- 0.00	Balance / 2
Split 3:	0.00	- 0.00	Balance / 3
Split 4:	0.00	- 0.00	Balance / 4
Split 5:	0.00	- 0.00	Balance / 5
Split 6:	0.00	- 0.00	Balance / 6
Split 7:	0.00	- 0.00	Balance / 7
Split 8:	0.00	- 0.00	Balance / 8
Split 9:	0.00	- 0.00	Balance / 9
Split 10:	0.00	- 0.00	Balance / 10
Split 11:	0.00	- 0.00	Balance / 11
Split 12:	0.00	- 0.00	Balance / 12

Ok Cancel

Figure 1. Setting up Variable Payments ([see larger image](#))

On a given row of the table, a range is created by the user. The divisor is the logical row number.

A Fixed Payment scenario is show in Figure 2.

Payroll Deduction Split Setup

Automatically Split Payroll Deductions:

All (Everyone pays split payroll deduction)
 None (Everyone pays their full balance)
 Some (Each person is set up individually)

When "Some" is selected, you have the option to globally change all customers to either "Yes" or "No." You can then selectively change individual customers using the customer maintenance window.

Set all customer to Yes Set all customers to No

Payment Type:

Fixed Payment is fixed, as defined by their balance.
 Variable Payment is calculated by dividing their balance by the number of splits that are defined.

	Current Balance		Payment
Split 1:	0.00	- 0.00	FULL
Split 2:	0.00	- 0.00	0.00
Split 3:	0.00	- 0.00	0.00
Split 4:	0.00	- 0.00	0.00
Split 5:	0.00	- 0.00	0.00
Split 6:	0.00	- 0.00	0.00
Split 7:	0.00	- 0.00	0.00
Split 8:	0.00	- 0.00	0.00
Split 9:	0.00	- 0.00	0.00
Split 10:	0.00	- 0.00	0.00
Split 11:	0.00	- 0.00	0.00
Split 12:	0.00	- 0.00	0.00

Ok Cancel

Figure 2. Setting up Fixed Payments ([see larger image](#))

In this case, the user must enter an amount to be deducted for each range of balances.

This window contains three arrays. The first column, range minima, is populated from `SPL:Min[x]`. The second column, range maxima, is populated from `SPL:Max[x]`. And the third is populated from `SPL:Payment[x]`.

The way this window operates is to hide or unhide alternate third columns. That is, if the selected Payment Type is "Variable," the entry fields are hidden and the string literals are unhidden. If the Payment Type is "Fixed," the string literals are hidden and the entry fields are unhidden.

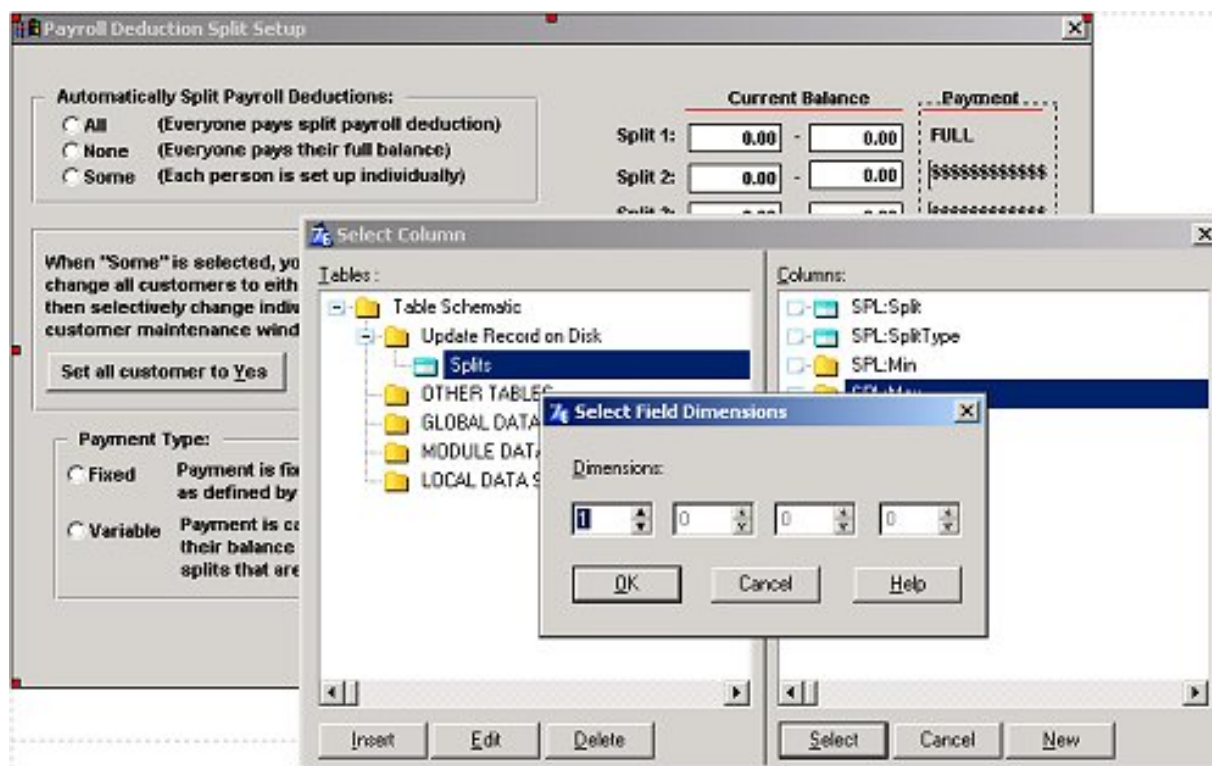
So, I end up with ranges defined by the user's entries in matching `SPL:Min[x]` and `SPL:Max[x]` fields. For example:

```
Split 1 = SPL:Min[1] to SPL:Max[1]
```

```
Split 2 = SPL:Min[2] to SPL:Max[2]
```

Etc.

Nothing remotely difficult so far. The IDE even assists the process of populating these fields by requiring the array element and providing entry controls when populating (see Figure 3).

**Figure 3. Selecting element number during population ([see larger image](#))**

My problems start when trying to protect users from themselves. Specifically,

- A range maximum must be greater than the range minimum;
- Each range's minimum value must be .01 greater than the previous range's maximum value (except the first, of course);
- If the Payment Type is "Fixed," the corresponding SPL:Payment[x] field cannot be zero.

Less Than, Greater Than

Ensuring that SPL:Max[1] is greater than SPL:Min[1] is not at all difficult. When SPL:Max[1] is accepted (doesn't make any sense to check before the range maximum is completed, does it?), if SPL:Min[1] => SPL:Max[1] then the range is not useable. In this case, either the two values are the same or they are reversed.

The full code for this check is:

```
If SPL:Min[1] => SPL:Max[1]
  Message('Range end point should be greater than ' &|
    'its beginning point.', 'Bad Range', ICON:Hand)
  Select(?-1)
End
```

Since SPL:Max[1] is the currently selected field, Select(?-1) backs the user up to SPL:Min[1] so the user can start over. ("?" indicates the current field.)

Now I need to generalize this code so that I don't have to type it into each of the 10 embeds, changing the subscripts. To accomplish this, in each SPL:Max[x], Accepted embed, I save the subscript number and call a common routine:

```
Spot = <split number>
Do CheckVector
```

This routine uses the "passed" number so that one codelet can handle any number of Max[x] fields I may have:

```
CheckVector Routine
If SPL:Min[Spot]
  If SPL:Min[Spot] => SPL:Max[Spot]
    Message('Range end point should be greater than its ' &|
      'beginning point.', 'Bad Range', ICON:Hand)
    Select(?-1)
  End
End
```

Note the If SPL:Min[Spot] check. This prevents the code from spuriously presenting the message if there is no range minimum, i.e., if there is no data entered for a specific range.

So, what's the problem?

The Problem Within "The Problem"

The two remaining checks get much more exciting.

The two checks I still have to make are: (1) that ranges are continuous (i.e., a new range begins at .01 more than the last range ended) and (2) that Fixed Payment schemes have a payment for each range created.

To ensure continuity, I modify the approach used for checking that $\text{Min}[x] < \text{Max}[x]$. In the Accepted embed for any range maximum:

```
Spot = <split number>
Do CheckRange
```

The common routine checks the previous range end ($\text{SPL}:\text{Max}[\text{Spot} - 1]$) against the entered value. If it is not $+0.01$, it messages and changes the value.

CheckRange Routine

```
If SPL:Min[Spot]
  If SPL:Min[Spot] <> SPL:Max[Spot-1] + .01
    Message('The previous range ended at ' &|
      Clip(Left(Format(SPL:Max[Spot-1],@n8.2))) &|
      ' the next range should begin at ' &|
      Clip(Left(Format(SPL:Max[Spot-1]+.01,@n8.2))) &|
      ' instead of ' &|
      Clip(Left(Format(SPL:Min[Spot],@n8.2))) &|
      '. Amount will be adjusted.', 'Invalid Data', ICON:Hand)
    If 0{Prop:AcceptAll}
      0{Prop:AcceptAll} = 0
    End
    SPL:Min[Spot] = SPL:Max[Spot-1] + .01
    Select(<something>)
  End
End
```

Note the `If SPL:Min[Spot]` condition. This is present so that the check will not execute when `If SPL:Min[Spot] = 0` during form completion (AcceptAll a/k/a Non-Stop Select mode). If I do not do this, every range would have to be completed (each zero `SPL:Min[Spot]` would be forced to equal `SPL:Max[Spot - 1] + .01` and each `SPL:Max[Spot]` would have to be greater than `SPL:Min[Spot]`— not good if the user only wants two or three ranges).

To ensure Fixed Payment schemes have payment amounts, the following code in `ThisWindow.TakeCompleted`, Before Parent call does the job:

```

If SPL:SplitType = 'F'
  Loop Spot = 2 to 12
    If SPL:Max[Spot] and ~SPL:Payment[Spot]
      Message('You have a range without a payment', 'Missing &
        Data', ICON:Hand)
      Select(<something>)
      Return Level:Notify
    End
  End
End
End

```

Please note the dummy `Select()` statements in both codelets. *This* is the problem. What is the correct argument for `Select()`?

In RangeCheck,

```
Select(?SPL:Min[Spot])
```

should select the first field in the new range.

In the check for a payment amount,

```
Select(?SPL:Max[Spot] + 1)
```

or

```
Select(?SPL:Payment[Spot])
```

should select the field after `SPL:Max[Spot]`, i.e., the payment field.

NOTE: If you are not familiar with the "?" notation or Field Equate Labels, look up "Field Equate Labels" in the on-line help. It is very important you be aware of this subject. I'll show you just how important in a moment.

Neither `Select()` statement will compile. Each produces an "Unknown Identifier" error.

Precisely what is going on here (but you know what I really said)?

The Language Reference Manual (a/k/a on-line help) explains it. Under the help for "Field Equate Labels," is the answer:

Field Equate Labels for USE variables which are array elements always begin with a question mark (?) followed by the name of the USE variable followed by an underscore and the array element number. For example, the field equate for `USE(ArrayField[1])` would be `?ArrayField_1`. Multi-dimensioned arrays are

treated similarly (?ArrayField_1_1, ?ArrayField_1_2, ...). You can override this default by explicitly specifying the Field Equate Label for use by each control in the third parameter to the controls' USE attribute.

Array elements, it seems, are not populated on windows as one might expect. They are not populated using subscript notation (despite what you see when you examine the generated code for a window containing arrayed variables).

Assignments are made using the variable with a subscript like:

```
SPL:Min[5] = SPL:Max[4] + .01
```

but window controls are addressed using a different syntax (see, I told you arrays were loathsome things) like:

```
?SPL:Min_5
```

If I were willing to forgo common routines or locally derived methods, I'm done. But, I am not willing to have to retype my checking code for each and every variable that needs it (29 in the current example; multiply by three for each additional range I populate).

Why is using a routine or local method a problem? It is because

```
Select(?SPL:Min_5)
```

can be interpreted, and interpreted correctly, by the compiler. `Select(?SPL:Min_Spot)`, where `Spot = 5`, cannot. This statement will not compile.

What's a Poor Programmer To Do?

I categorically refuse to write an error check for each control that might require it. Yet it appears that I cannot use my variable, `Spot`, in the common routine's `Select()` statements. But, to use common routines, I *must* use a variable.

The answer, provided by the Clarioneers listed above, lies in the last sentence of the specification for "Field Equate Labels," quoted above:

You can override this default by explicitly specifying the Field Equate Label for use by each control in the third parameter to the controls' USE attribute.

I can override the Field Equate Label. I can change Use's defaults, all of them, according to this. When I look at the specification for "Use," I find that this is very much the case (emphasis added):

USE Specifies a variable or field equate label.

label A field equate label to reference the control or structure in executable code. This must begin with a question mark (?) and meet all the requirements of a valid Clarion label....

number An integer constant that specifies the number the compiler equates to the field equate label for the control (PROP:Feq, equivalent to {PROP:USE,2}).

equate A field equate label to reference the control in executable code when the named variable has already been used in the same structure. This provides a mechanism to provide a unique field equate when the variable would not.

The USE attribute (PROP:USE) specifies a field equate label for the control or structure, or a variable for the control to update.

USE with a label parameter simply provides a mechanism for executable source code statements to reference the control or structure....*The USE attribute's number parameter allows you to specify the actual field number the compiler assigns to the control.* This number also is used as the new starting point for subsequent field numbering for controls without a number parameter in their USE attribute. Subsequent controls without a number parameter in their USE attribute are incremented (or decremented) relative to the last number assigned.

According to this, I can assign a Use label of my choosing. More than that, I can assign the actual control number the compiler will use to identify this control (taking care, of course, to specify a number high enough that it cannot be used by the compiler itself).

And, this can be done from inside the window formatter:

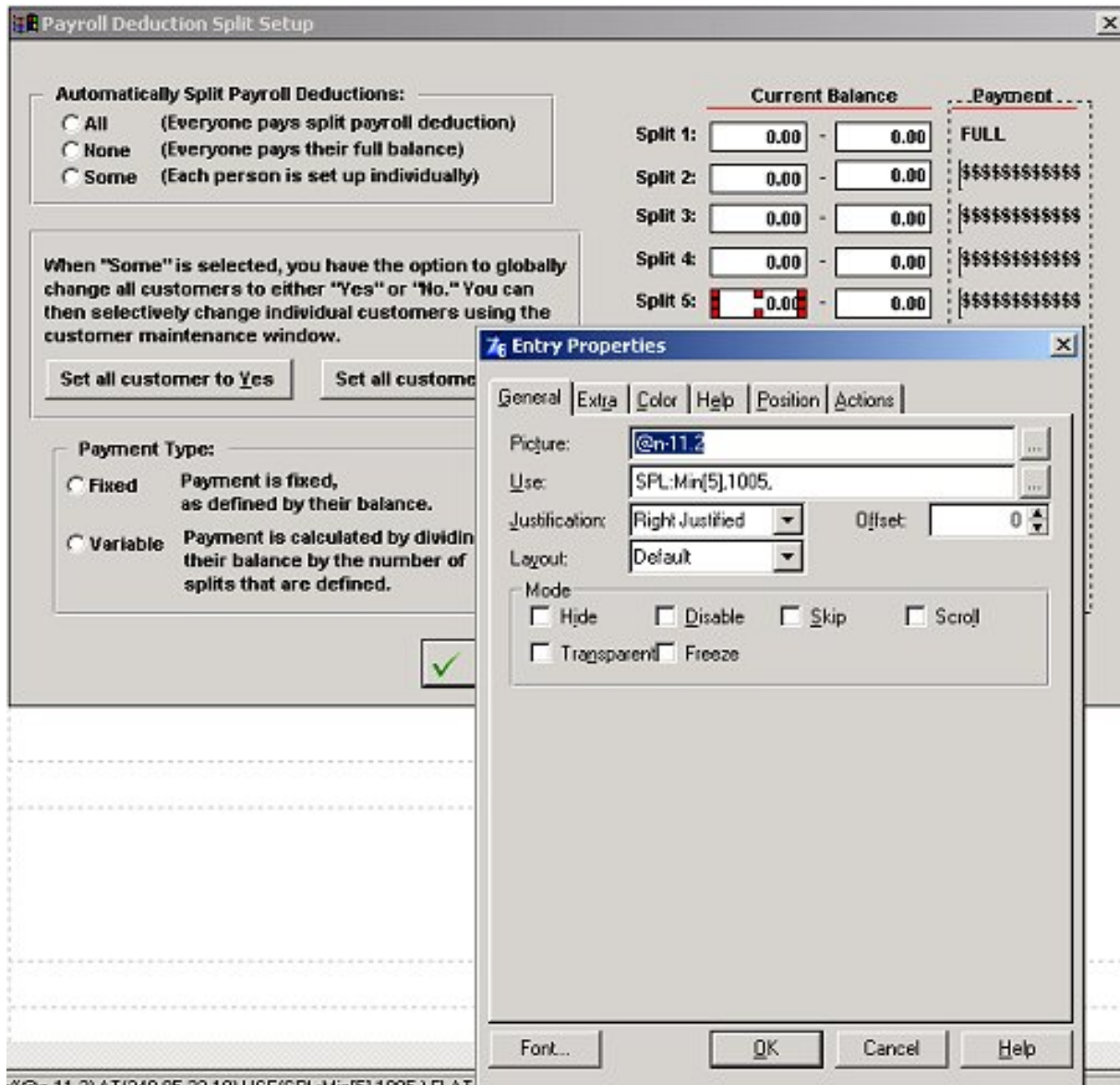


Figure 4: Specifying a Field Equate in the IDE ([see larger image](#))

Bingo!

If I can tell the compiler the Field Equate (the actual FEQ, not "just" the Field Equate *Label*) then I can create a simple algorithm to determine which control needs to be selected.

Suppose I force all the controls in the minimum column to use FEQs starting with 1000, 2000 for the second column and 3000 for the third column. Then, any control in the minimum column is correctly identified by $1000 + \text{Spot}$. The maximum column's controls are uniquely identified by $2000 + \text{Spot}$ and the payment column's by $3000 + \text{Spot}$.

My `Select()` statements will be:

```
Select(1000 + Spot)
Select(2000 + Spot)
```

or

```
Select(3000 + Spot)
```

and the correct control will receive input focus. (Note the "?" is no longer appropriate here; I am using the FEQ directly, not a compiler equate as I would under normal circumstances.)

Just what the doctor needed.

But wait! There's more!

I don't know why, but controls from `SPL:Min[4]` and up didn't generate `Event:Accepted` (I suspect my embedded code not executing impeded the solution a bit). But, with complete control over the Field Equates, this was no longer a problem (my attitude toward things that appear to be bugs – some really are bugs, some are my own stupidity and some are "great mysteries of the universe" – and the attitude of most Clarioneers, has always been "if I can work around it, I don't have the time to wait for SoftVelocity to fix it").

Knowing the FEQ (without having to count controls in the window structure and without a screen formatter that tells me what they are, as the CDD screen painter did), I can easily test whether one of my controls was completed.

In `TakeEvent` (any priority, but I use 1), I can do my own test:

```
Case Accepted( )
Of 1002
  Spot = 2
  Do CheckRange
Of 1003
  Spot = 3
  Do CheckRange
Of 1004
  Spot = 4
  Do CheckRange
Etc.
```

Also, now that I have a working procedure, some alternatives occur to me. (Of course, I have not

Years ago, I saw the C code for a simple window. This code required an include file enumerating each control on the window and assigning it a number. Code intended to affect a control had to use this number.

This number is the "FEQ," the Field Equate.

In Clarion, we have "Field Equate Labels" assigned by the compiler. Field Equate Labels begin with a question mark and Clarion developers can use them or the actual FEQ to affect a control. For example,

```
?Prompt27{Prompt:Text} =
`This is a prompt`
```

or

```
153{Prompt:Text} = `This is
a prompt`
```

The big advantage of Field Equate Labels is that when you add or remove controls, the compiler will adjust the Field Equate automatically. You will not have to recount the fields.

I did get a good laugh out of the C code

actually tried to implement any alternatives. After all, I have a *working* procedure.)

A local method, instead of the routines I use, would probably have worked with minimal additional typing (the main reason I insisted on keeping my routines).

I could easily create two local methods (assuming I got `Event : Accepted` working):

```
CheckVector(Long pSpot),Byte
```

And

```
CheckRange(Long pSpot),Byte
```

The code for these two methods would be substantially the same as the routines except:

1. "Spot" is changed to "pSpot" (I'm just more comfortable passing parameters, otherwise, there's no special reason to do so)
2. the `Select()` statement would be removed from the codelet

and

3. if the check passes, I return "1" and if it fails, I return "0"

For example,

```
If SPL:Min[pSpot]
  If SPL:Min[pSpot] => SPL:Max[pSpot]
    Message('Range end point should be greater than its ' &|
            'beginning point.', 'Bad Range',ICON:Hand)
    Return 0
  End
  Return 1
End
```

My embedded code becomes:

```
If ~CheckVector(<split number>)
  Select(?SPL:Max_<split number> - 1) !use standard Equate Label
End
```

`CheckRange` and its call could be similarly handled. These would be close enough to my time-honored use of routines to be acceptable, possibly even a bit more elegant.

Another thing that reading the Language Reference Manual implies is that I could store the real FEQ of the current control and use it in my `Select()`. For example, for element number 54:

```
Spot = 54  
ThisFEQ = ?SPL:Min_54{Prop:FEQ}  
Do CheckRange
```

ThisFEQ, a new local variable (a Long), contains the current FEQ. ThisFEQ can then be used in Select () statements (without the "?").

Again, this would not involve enough extra typing to disqualify using this approach. And, using Prop:FEQ appears to be easier to use in the TakeCompleted loop I do to check the SPL:Payment[x] fields.

Summary

I think I have made my case: arrays are the work of the devil.

Where a normal variable has a Use variable of a question mark plus the variable Label, an array behaves normally for assignments but takes a convoluted form for its Use variable. And, those Use variables do not work well in generalized code where they must be identified by variables.

But, with a big hand from a group of very helpful folks, you learn things about "Field Equate Labels" and "Use" that you never knew. This knowledge makes array elements almost as accessible as a normal variable.

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

Reader Comments

[Add a comment](#)

[Steve, In some of my applications, I find I need to...](#)

Clarion Magazine

online sales and delivery
for your applications & tools

Developer **PLUS**

Clarion Books

[Pre-Order](#) the Clarion Tips & Techniques book and **SAVE \$20!**

Update! January 20, 2004 We're almost done! Indexing is now complete, and the *Clarion Tips and Techniques* book is getting a final proofreading. We're hoping to start shipping around the end of January. You can now read both the table of contents and the index online.

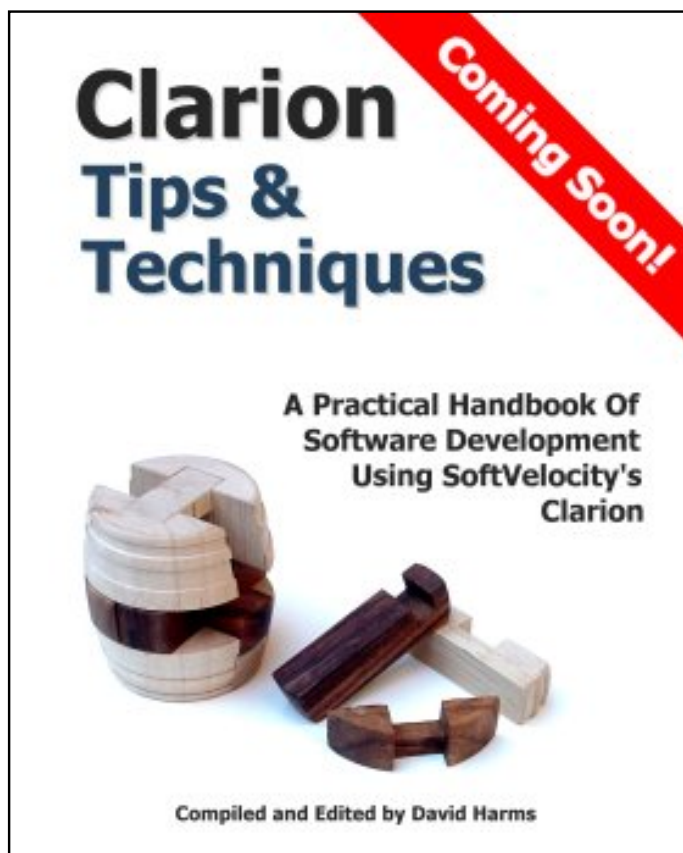
[View the table of contents \(PDF\)](#)

[View the index \(PDF\)](#)

About the book

Clarion Tips and Techniques is a book of approximately 630 pages, containing selected articles from Clarion Magazine's first five years in publication, as well as a small number of articles originally published in Clarion Online. These articles were written for Clarion 5.0 and 5.5, but many are just as applicable to Clarion 6. The list price is \$89.95, but if you [pre-order now](#), you can get it for **just \$69.95**, a savings of \$20. This offer will end once the book is in print. So get your order in now!

[Pre-Order](#) your copy of *Clarion Tips and Techniques* now for **just \$69.95!**



Subscription Not Required: The current purchase process requires you to **log in** if you have an **existing ClarionMag user ID**, or **register a new (free) ID**. Apparently this has given

some people the impression that you need to be a subscriber to buy the book. This is most emphatically *not* the case! I need your address in order to calculate freight and ship you the book, and the easiest way to do this is to simply have you use the existing means of registering with Clarion Magazine, which has the added benefit of giving you access to certain [free articles](#), as well as making it easy for you to sign up for our double opt-in mailing lists. I regret any confusion this may have caused - if it appears this will be a problem I'll certainly look at amending the order process so you don't have to register a user id.

Multiple Copies: Several people have asked me how to purchase more than one copy of the book at one time. Until such time as I've made the necessary changes to the shopping cart, you can use these links: [order 2 copies](#) - [order 3 copies](#) - [order 4 copies](#) - [order 5 copies](#). The shopping cart will still indicate that the book has been added and there is one item in the cart, but when you go to the checkout or to edit the cart you'll see that the quantity is as specified.

Refund policy: You can, of course, get a full refund on your book order at any time up until the book ships, and if you are not satisfied with the book you have 30 days from shipment date to return the book, in new condition, to CoveComm Inc, publisher of Clarion Magazine, for a full refund or credit of the book purchase price.

The Proof Copy

The first proof copy is back from the printer! Click on the images below for larger pictures. We still have some changes to make - the cover needs some tweaking, and of course we have to complete the index, so the book isn't shipping yet. We expect to finish production and begin shipping in January.



To be reminded when the book ships, enter your email address below. And check this page regularly for updates, including an upcoming table of contents and a PDF excerpt!

We are using a major US printing house to produce these perfect bound, soft cover books. As you can see from the pictures above, this is a book just like you'd expect to find on the shelves at your local computer bookstore.

Notify me when this book is shipping!

Email address:

Why is Clarion Magazine printing books?

Clarion Magazine, the online publication, is very popular with Clarion developers, and we're proud of the massive library of information we've published in HTML and PDF form. So why are we now printing books? Because as handy as HTML and PDF documents are, for many of us they still aren't as convenient and, well, enjoyable to read as a printed, bound book.

We are continuing to convert Clarion Magazine's HTML pages to book form. There is a stunning amount of material to choose from; in a 7.5" x 9.25" book format, the Clarion Magazine web site contains some 6500 pages of Clarion articles. That's about two feet of shelf space! If there is sufficient interest we will consider publishing all of this material, but it would be impossible to bring it all to press in a short period of time. As a result, we're focusing our efforts on areas likely to be of most immediate interest to Clarion developers. Stay tuned for more book news! And if you have any questions just send us an [email](#).

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.