# Clarion Magazine

This edition includes all articles, news items and blog posts from June 1 2011 to June 30 2011.

## Clarion News

Read 20 Clarion news items.

## The ClarionMag Blog

Read 1 blog entry.

## Articles

### Clarion International Developers Conference (CIDC 2011) To Be Held In Orlando FL

June 10 2011

The 2011 Clarion International Developers' Conference will be held in Orlando, Florida on September 26th - October 1st. This is the official Clarion DevCon for 2011, presented by ClarionLive and SoftVelocity.

The conference venue is the Rosen Centre Hotel in Orlando. This is a six day event: three days of talks, and three days of workshops (separate registration available).

### Clarion 8 Beta Notes for June 2011/Open Discussion

June 11 2011

Notes on the Clarion 8 beta for June, 2011, posted as comments to this article. Subscriber comments are also welcome.

### Tip of the Week: Copying Your Tools Menu

June 12 2011

Copying your tools menu items between versions of Clarion is as easy as copying a file.

## Tip of the Week: Hiding Compile Warnings

June 13 2011

Sometimes, when you're doing a large batch compile, you only want to see errors and not warnings.

## Time and Time Again, Again

June 16 2011

Failure to understand the data, Dr. Parker points out, inevitably results in a lot more work than is necessary. An analysis of the date/time datum yields an interesting change in strategy, and much cleaner code.

## Tip of the Week: Two Ways To Compile

June 17 2011

There are two ways to compile an app without generating: If you have circular dependencies, one way will give you an error and the other will not.

## ReplaceText

June 23 2011

In a recent article, Steve Parker alluded to Peter Hermansen's "replace text" function. There were questions. Steve has answers.

## Tip of the Week: Multiple Installs

June 24 2011

Keeping separate installs for every version of Clarion can make it much easier revert to an earlier build.

## Understanding Field History

June 28 2011

How can you tell if a specific field has changed? How can you know the previous value? How can you know if the record has changed. The Steve knows.

## GainFocus: When and How

June 29 2011

GainFocus has had some aspersions cast on its character. Does it work reliably? Dr. Parker puts on his deerstalker and goes in search of the truth. Or something.

## Tip of the Week: Multiple Installs for Team Development

June 30 2011

If you're in a shop where multiple developers work from one server, here's how you can manage multiple Clarion installs with minimum team disruption.

## Testing Classes, Creating Classes

June 30 2011

Having succumbed to writing classes, Steve Parker now finds himself thinking about how to test those classes. And his testing leads him to helpful way of creating new methods.

# Clarion News

## Clarion 8 Build 8399

Clarion 8 build 8399 has been released to CSP participants. According to the release notes, the embed tree issues have been fixed.

*Posted June 2 2011 (permanent link)*

## Clarion 8.0.8421

Clarion 8 build 8421 is now available to CSP participants. This release improves the button refresh situation in the app tree, fixes some GDI leaks, and corrects a number of other issues.

*Posted June 10 2011 (permanent link)*

## LexisNexis Launches HPCC Platform For Big Data: The Clarion Connection

LexisNexis has released an open source platform for high performance massively parallel data processing. Former Clarion/Topspeed folk have a prominent role: development team members include David Bayliss, Richard Chapman, Gavin Halliday, and Jake and Gordon Smith. Documentation/training team members include Jim Defabia, Richard Taylor, and most recently Bob Foreman.

*Posted June 15 2011 (permanent link)*

## ProScan 3.0 Released - New features and support for Clarion 8

LANSRAD announces the immediate availability of version 3.0 of Clarion ProScan. This is a new version of the product that includes new features, enhancements and support for Clarion 8. This is a free update if you purchased ProScan (or the upgrade to Version 2.x) in the last 12 months. New features include: Merge pages from one TIFF file into another without scanning; Merge pages from a FAX document saved as a TIFF file into any page position of the TIFF loaded in the Scan Manager; Advanced Print Control dialog; Advanced Export to PDF Control dialog; Conditional generation of the class declarations for the Path and Registry classes on every extension, control and code template.

*Posted June 23 2011 (permanent link)*

## ProImage 3.0 Released - New features and support for Clarion 8

LANSRAD announces the immediate availability of version 3.0 of Clarion ProImage. This is a new version of the product that includes new features, enhancements and support for Clarion 8. This is a free update if you purchased ProImage (or the upgrade to Version 2.x) in the last 12 months. New features include: Load From TIFF feature allows you to open a external TIFF file and page through it with a built-in a TIFF viewer; Load from FAX feature allows you to open a external FAX TIFF file and page through it with a built-in a TIFF viewer; Advanced Full Page Previewer, designed for developers who own both ProImage 3.0 and ProScan 3.0, allows you to send an ImageEx bitmap or an external file into the previewer for full page viewing; Conditional generation of the class declarations for the Path

and Registry classes on every extension, control and code template.

Posted June 23 2011 (permanent link)

## Build Automator Adds C8 Support

The latest release of Build Automator has full support for compiling Clarion 8 solutions and projects and also fully supports code generation for both Clarion 8 solutions and apps.

Posted June 23 2011 (permanent link)

## DMC 2.4.0.1425

DMC 2.4.0.1425 is now available. All demos are Clarion 8 compatible (installers) and contain separate C7 demo apps.

Posted June 23 2011 (permanent link)

## Whitemarsh June 2011 Newsletter

Whitemarsh updates for June include: A new version of the Free Metabase System; Documents that describe the Metabase System; Three New Metabase System Modules; New Short Papers; Finalizing ANSI SQL 2012 Standard. Whitemarsh is focused on implemented-completely-and-correctly-the-first-time approach to database and business information system development.

Posted June 23 2011 (permanent link)

## Tracker Products Updated

Tracker products Clarion Version: 4.0195 (updated on 7 Jun, 2011) are now available. Be sure to choose the Clarion downloads.

Posted June 23 2011 (permanent link)

## SysTrack Open Sourced

SysTrack is a Clarion wrapper for the trackbar win32 common control.

Posted June 23 2011 (permanent link)

## SysList Open Sourced

SysList is a Clarion wrapper for the Win32 listview common control.

Posted June 23 2011 (permanent link)

## Systree Open Sourced

SysTree is a Clarion wrapper for the Win32 tree-view control.

Posted June 23 2011 (permanent link)

## SysDTP Open Sourced

SysDTP is a Clarion wrapper for the date time picker win32 common control.

Posted June 23 2011 (permanent link)

## SysMonthCal Open Sourced

SysMonthCal is a Clarion wrapper for the month calendar win32 common control.

Posted June 23 2011 (permanent link)

## ImageEx Clarion 8 Compatible

A Clarion 8 compatible version of ImageEx has been released. This is a $49 upgrade, unless you purchased within the last 12 months in which case the upgrade is free.

*Posted June 23 2011 (permanent link)*

## RichReport Clarion 8 Compatible

A Clarion 8 compatible version of RichReport has been released. This is a $29 upgrade, unless you purchased within the last 12 months in which case the upgrade is free.

*Posted June 23 2011 (permanent link)*

## J-Zip for C8

J-Zip is available for Clarion 8.

*Posted June 23 2011 (permanent link)*

## Build Automator 1.70.1277

Build Automator version 1.70.1277 is available for download. Note that you must have a valid Maintenance plan to be able to install this build.

*Posted June 23 2011 (permanent link)*

## Clarion 8 Versions of Sterling Data Templates

To download the Clarion 8 versions please use the same download link and passwords as the Clarion 7 link.

*Posted June 23 2011 (permanent link)*

## DevCon Early Bird Registration Ends June 30

The Clarion International Developers Conference takes place in Orlando Florida from September 26th - October 1st. Three days of Conference Talks, followed by three days of Workshops. Early bird registration ends June 30, 2011.

*Posted June 23 2011 (permanent link)*

# The ClarionMag Blog

## Is CapeSoft giving away TWO iPad 2s?

A little birdie has whispered in my ear that CapeSoft are planning to give away not one, but TWO iPads at this year's Clarion International Developer's Conference in Florida. Apprently there are at least two white Apple boxes with "CIDC 2011" written on them floating around the CapeSoft office.

When contacted directly CapeSoft responded with "no comment".

Time will tell if this is indeed the case, but remember you read it here first on ClarionMag.

*Posted June 30 2011 (permanent link)*

# Clarion 8 Beta Notes for June 2011/Open Discussion

Posted June 11 2011

This is an open article for Clarion 8 early release discussion during the month of June, 2011. I'll post about releases, newsgroup comments, and other relevant issues. Subscriber comments are welcome!

## Article comments

*by Dave Harms on June 11 2011 (comment link)*

Build 8421 is out. Among other things the AppGen button flicker looks to be fixed, which is good news.

There have been a few new issues reported in the newsgroups. There seems to be a problem with icons, where the wrong icon size is chosen or the icon is not resized.

Checkbox properties are not showing in the data pad, at least under some circumstances. That's a showstopper for some.

*by Russell Eggen on June 11 2011 (comment link)*

The orphaned embed issue is fixed too. On the downside, if you have checkbox controls in your app, the properties pad is blank. Also, the dct editor has some issues when adding new entities. Pressing ENTER to save will not save your changes, IOW - the default values are used instead. Clicking on Ok does not suffer from this problem. But be careful on adding anything new there.

I've not found any showstoppers, but there are quirks in C8 that are not in C7.

Russ Eggen

*by Dave Harms on June 13 2011 (comment link)*

Another reminder that the C7 IP driver, Dynamic File driver and In-Memory file drivers are all compatible with C8 (as pointed out recently by Bruce Johnson). Just run the installer again and point it at C8.

*by Dave Harms on June 18 2011 (comment link)*

There was a very lengthy discussion on the IDE's ability to permanently delete files from disk, with several posters strongly opposed. As of release 8461 deleted files now go to the recycle bin, so hopefully that's good enough for everyone.

There's also been a fair amount of traffic about images posted by people using MesNews. Those

images come through as Part1, Part2 etc. I never used to be able to read these with Thunderbird, but lately they're displaying just fine.

There are some reports of duplicate symbol errors appearing in 8461, also one report of random unresolved external errors on DCTINIT@F.

---

*by Jimmy on June 19 2011 (comment link)*

Hi, I compiled one of my large apps with the previous release of C8 and apart from the checkbox property not showing, it compiled ok The new version compiling the same app previously in C8 now gives the following, which is a showstopper for me Duplicate symbol: _trflushblock@FP14internalstatePcUli in TREES.OBJ, TREES.OBJ Duplicate symbol: _tralign@FP14internalstate in TREES.OBJ, TREES.OBJ Duplicate symbol: _trstoredblock@FP14internalstatePcUli in TREES.OBJ, TREES.OBJ Duplicate symbol: _trinit@FP14internalstate in TREES.OBJ, TREES.OBJ Duplicate symbol: _zcalloc@FPvUiUi in ZUTIL.OBJ, ZUTIL.OBJ

I have tried to find where the boob is but the size of the app makes it a quest for the holy grail. I tried to imnport from the original app into a new app and get GPF's when I import any module with perceived fields no found!

Oh well back to 7.3 until it gets right - good stuff waiting though!

---

*by Dave Harms on June 20 2011 (comment link)*

Jimmy, there's been some discussion of this in the newsgroup. It appears SV is adding PNG support and there's a conflict between SV's libraries and those used by third party products such as Insight Graphing and ImageEx. Paul Blais reports that removing the Report to PDF template is a workaround, as is compiling in DLL mode instead of LIB mode.

Charles Edmonds has suggested that as long as you set the output for your DLL to DLL and you set the link mode to DLL you will not have a problem.

---

*by Dave Harms on June 20 2011 (comment link)*

There's a problem with adding a variable in the data pad using the + icon while the window designer is open, as reported by Brian B. The variable disappears when you exit the procedure. You can add variables if the window designer is not open.

---

*by Dave Harms on June 20 2011 (comment link)*

Abe Jimenez reported a problem in 8461 with radio buttons, confirmed by SV. Drag a data pad item with three must be in list values to the window, and the radio buttons are not populated in the option box.

---

*by Dave Harms on June 23 2011 (comment link)*

Another "Is Clarion 8 ready for production" thread, with the nearly unanimous answer of yes. C8 is a clear improvement over C7 (as it should be). And if it's good enough for Robert Paresi...

Here's one for you road warriors. As of 8461 please note that you cannot use 256 color mode with the IDE - the AppGen will crash. The only reason you'd want to use 8 bit color is bandwidth - it's a pretty ugly experience.

CapeSoft will be giving away a free iPad 2 at this year's DevCon. And remember, early bird registration ends June 30.

---

*by Dave Harms on June 23 2011 (comment link)*

Diego has reported that in the next release pressing Ctrl-M no longer hangs the IDE.

---

*by Russell Eggen on June 23 2011 (comment link)*

Found an obscure bug in 8461(exists in C7 too) that when you check the global app option to enable Exception Message with Procedure Stack, this has a side effect of the first and last row in a browse to duplicate. Uncheck this option and browse procedures behave as expected. This is reported fixed in the next release.

Russ

---

*by Dave Harms on June 23 2011 (comment link)*

That is obscure<g> - thanks Russ!

---

*by John Welch on June 28 2011 (comment link)*

I have found that the extremely long load times for the debugger in C7.3 do not exist in C8. The debugger and all symbols load nearly instantly; thus making debugging viable again.

BACK TO TOP

# Tip of the Week: Copying Your Tools Menu

## By Dave Harms

Posted June 12 2011

I have a bunch of items under my Tools menu in Clarion 7, and when I installed Clarion 8 it didn't pick up those items automatically. I don't know if the installer just doesn't do that, or if it has something to do with how I install Clarion (each version in its own directory). In any case, copying across Tools items is easy to do.

Tools menu entries are stored in a configuration file called Clarion-tools.xml. To migrate your tools settings, all you need to do is copy this file from:

%appdata%\SoftVelocity\Clarion\7.0

to

%appdata%\SoftVelocity\Clarion\8.0

If you've made any changes to your tools settings in Clarion 8 you may want to use a comparison tool like Beyond Compare to merge those changes.

---

*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), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).*

## Article comments

*by david griffiths on June 30 2011 (comment link)*

I would be interested in what tools you find useful in your tools menu.

BACK TO TOP

# Tip of the Week: Hiding Compile Warnings

## By Dave Harms

Posted June 13 2011

I periodically do a batch compile on a solution containing close to 200 APPs. And because some of this code is very old and written by many different developers, it isn't as clean as it might be. In particular there are a number of number of duplicate symbol warnings that appear during a batch compile. Work is being done to reduce that number, but in the meanwhile, when I'm doing a batch compile, all I really care about are the errors.

I do two things to make it easier to see the errors while the batch compile is happening. First, I grab the errors window by its tab and I drag it so it's undocked. If it's docked, and it's in a pad with the output window, the output window frequently grabs focus and I can't see the errors. With a lot of compiles there's just too much flickering and too much waiting when I click on the errors tab. When the errors window is on its own, it's always visible.

So that's step 1. Step 2 is only display errors.

Figure 1 shows the errors window in its default configuration.

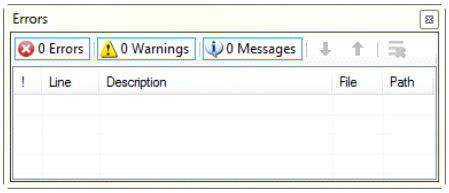


Figure 1. Default errors window

In Figure 1, Errors, Warnings and Messages are all latched buttons. In their default on state they allow the display of (as you'd expect) errors, warnings and messages. To change the state of the buttons you click on them once or twice. If the window is not yet selected, the first click will select the window and the second click changes the button state. If the window is already selected, then you only need to click once on a button to change its state. But be careful - the visual feedback on these buttons isn't great. I find it easiest to move my mouse off the button after clicking. When a button is disabled it doesn't look like a button at all. Figure 2 shows the Warnings button disabled, so only errors and messages display

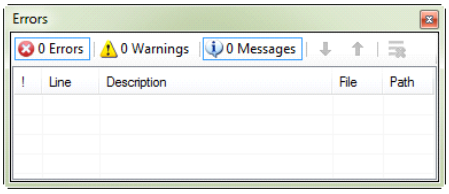


Figure 2. Disabling the display of warnings.

If you have a big batch compile to do, you'll probably want to set the errors window up beforehand. It's much more difficult to do in the middle of a batch compile as the IDE is sometimes unresponsive.

---

*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), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).*

## Article comments

BACK TO TOP

# Time and Time Again, Again

By Steven Parker

Posted June 16 2011

In "Time and Time Again," I experienced difficulties computing the time between two StarDates when there had been a midnight rollover. While I finally did create code that functioned both correctly and reliably, I was unhappy with the solution.

I speculated in that article that I was committed to using my newly created and tested time calculating class to the exclusion of rationality. But, somewhat more fundamental, I realized, was that I insisted on applying previously developed time methods to StarDates without completely appreciating the tool I had in a StarDate.

Failing to understand the data (and a StarDate is *just* a datum) inevitably results in a lot more work than is necessary. Additional work, usually accompanied by an uneasy feeling about the code, is the best case scenario. Assuming you get the code working, failure to understand your basic data often leads to unreliable code. That, I think, may be worse than code that doesn't work at all. At least there is no temptation to put non-functioning code into production.

## The Nature of the Beast

What is the essence of a StarDate?

In short, a StarDate is a completely self-contained description of a point in time. It is a total description of a date and a time on that date, precise to 1/100 of a second. A StarDate is unique and definitive of a 1/100 second "time slice."

That means a StarDate is convertible to a Clarion Standard Time. It is convertible to an integer, an integer expressing *everything* there is to know about the date-time. Granted, converting a date and time to Clarion Standard Time results in a number larger than that permitted by Clarion for times. 8640000 (one day expressed as clock ticks) plus *any* time whatsoever ("1" being the minimum) *must* exceed the maximum permitted value for a time.

However, none of this means that the resulting number is not "Clarion Standard." It means that it is not something Clarion can recognize (really, "display") as a "time (of day)." That's all. Just because a number is big, *really* big, does not mean it stops obeying the transformation rules of Clarion Standard Time. It just needs a little massaging.

A number, call it "RBN" (for "*r*eally *b*ig *n*umber," pronounced, I think, "RiBboN") divided by 100 gives the number of seconds represented in the number, however many there may be. RBN, no matter how small or how large, divided by 6000 still returns the number of minutes represented in the number, even if a huge number of them. And `RBN % 6000` still returns the seconds component. Size does not matter.

All I need to do is translate my StarDate (or separate date and time data) into integers containing the total clock ticks.

## Making Really Big Numbers

I'm going to start by working with discrete date and time fields. I'll take up transforming StarDates afterwards.

The idea is to "change" a date (not the number of days but the actual Clarion date) into Clarion Standard Time ticks. Then, I need to add that result to the time variable or component to get my RBN.

Turning a date into time ticks is just a simple multiplication:

```
DateVar * 8640000
```

Next, add the known time to the date-as-ticks computation:

```
RBN = StarTime +  (StarDay * eDay)
```

This *should* do the job perfectly, except that it fails.

In the demo app, from the main menu, Tests | Make / Parse RNB (Longs). Enter a time (the date defaults to `Today()`) and you will see something like Figure 1.

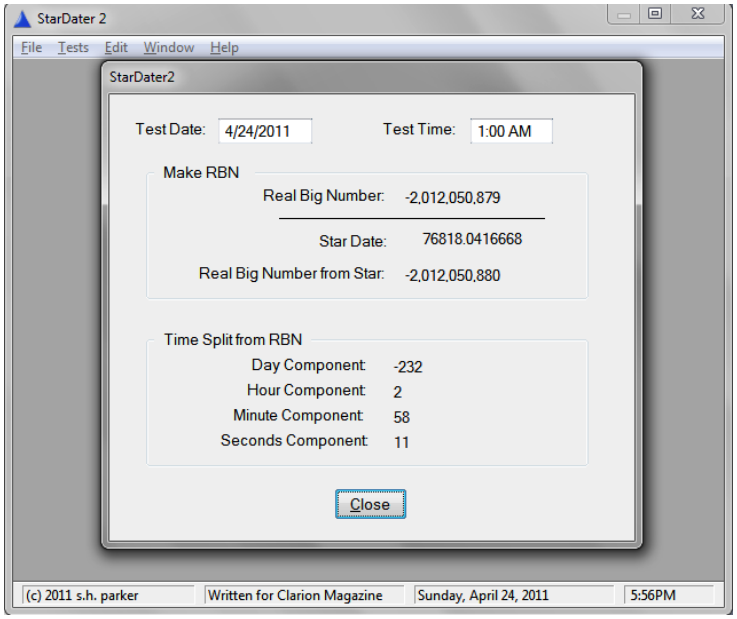


Figure 1: Making a RBN with Longs

Note how the computation is returning a negative number. This is impossible. Negative time is *prima facie* evidence of an error.

Of course, there is one circumstance in which negative time is possible: When the result exceeds the upper range of a Long. The remaining calculations, all based on algorithms original described in "Marking Time, Part 1," confirm that I must have wrapped past the maximum value for a `Long`.

Clarion's automatic data conversion, on which I so often relied in my last few articles, sees three integers and uses Longs. But if you manually multiply 76818 (the Clarion date on which I did the screen

capture) by eDay, you end up with a number well beyond the upper range of a Long. (Actually, it also exceeds the upper limit of a ULong, by a substantial amount, too.)

Loading the incoming parameters, which are Longs, to locally declared Decimals, that is, casting the Longs to Decimals, fixes the problem. In the demo app, Tests | Make / Parse RBN (Figure 2).

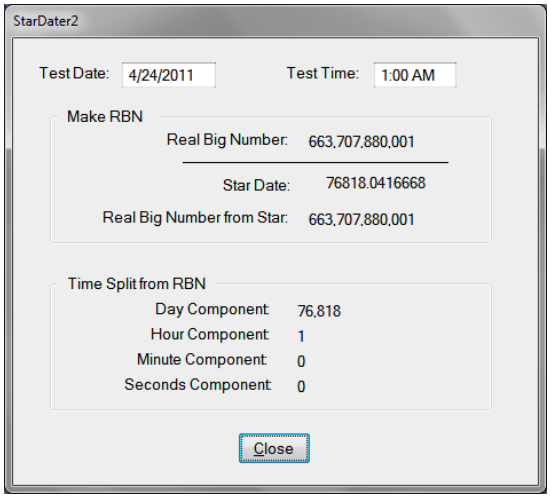


Figure 2: Casting Longs before the Decimals

(Feel free to ignore the unexpected "Days" value. The days computation isn't designed for single date computations and hasn't been debugged yet.)

This is the code to accomplish making a RBN from a date and a time variable:

```
ThisWindow.RBNfromDT PROCEDURE(Long pDate,Long pTime)

RetVal  Decimal(20)
D1      Decimal(20)
T1      Decimal(20)

  CODE
  D1 = pDate            ! cast to Decimal
  T1 = pTime
  Return T1 + (D1 * eDay)
```

The variable receiving the return value, obviously, is also a Decimal(20).

## Making Really Big Numbers from a StarDate

Knowing that Longs are inadequate to the purpose and knowing that I have a method to parse a StarDate into its date and time components makes composing a RBN from a StarDate fairly straightforward (i.e., very little code).

Receive the StarDate into the procedure. Notice that, this time, I have used a runtime defined parameter to accommodate both Decimal and Real StarDates.

Once I have the date and time reconstituted, using the ParseStarDate method previously described, I can call the method just defined to create the RBN.

```
ThisWindow.RBNFromStarD PROCEDURE(? pStar1)

D1  Long
T1  Long
R1  Decimal(20)
TempDate DECIMAL(15,8)

  CODE
  TempDate = pStar1
  MyStarDate.ParseStarDate(TempDate,D1,T1) ! reconstitute
  T1 += 1                  ! add back the   midnight   tick
  R1 = Self.RBNfromDT(D1,T1)           ! call new method
  Return R1
```

## FNSplit for Times

In the last article, it occurred to me that all my time difference computations were based on using two times and computing the difference, then extracting the components in readable form.

But, the core code in all the methods still operate on a single variable, the variable holding the difference, and extracting the components.

Time to stop thinking about “two variables.”

```
Seconds = Difference  / 100
```

So, if I have the difference between two StarDates, I can convert that to an RBN. Or, I can convert two StarDates to RBNs and subtract RBNs. I can pass the “difference RBN” to a procedure that extracts the days, hours, minutes and seconds (I’ve given up worrying about hundredths of a second at this point).

This should eliminate the additional steps and computations I had to do in my `StarET` method.

Where `pTime` is an RBN difference:

```
Days    = INT(pTime / eDay)
hrs     = pTime % eDay        ! non-"day" portion
Hours   = INT(hrs / eHour)
 If Hours > 24                ! never see this
   stop('Too many hours!')
 End
MinSec  = pTime % eHour       ! non-hours portion
Minutes = INT(MinSec / eMinute)
Secs    = MinSec % eMinute    ! non-minutes portion
Seconds = Secs / eSecond

pDays = Days                  ! set up return values
pHours = Hours
pMinutes = Minutes
```

```
pSeconds = Seconds
```

This code is much more readable (and, therefore, easier to maintain or, perish the thought, debug) than the `StarET` method presented last time.

And, most importantly, it works as expected (with an occasional one tick discrepancy, though I think I've found them all).

In the demo app, select the Compare Dates item:

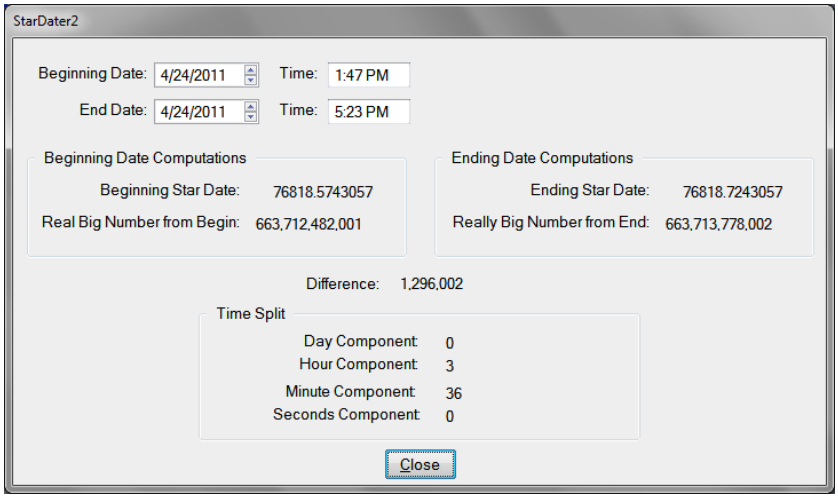


Figure 3: Same day time difference

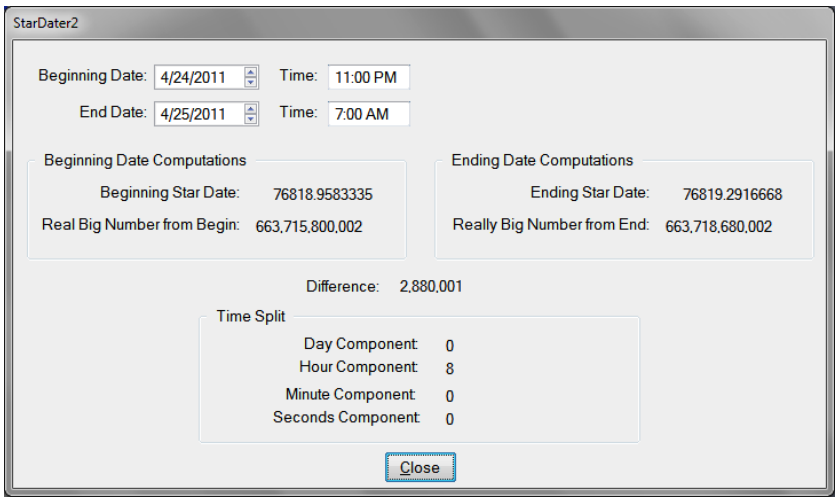


Figure 4: Midnight rollover but less than 24 hours total

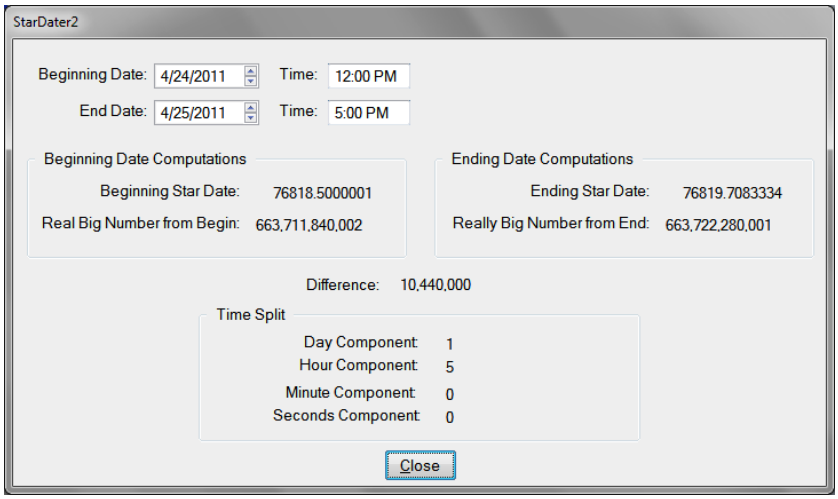


Figure 5: Midnight rollover, more than 24 hours

The actual calls are as follows. When both beginning date and time are entered:

```
If BeginDate and BeginTime
  MyStarDate.MakeStarDate(BeginDate,BeginTime, |
    BeginStarDate)
  RBNBeginDate = Self.RBNFromStarD(BeginStarDate)
  ThisWindow.Reset(1)
End
```

I make a StarDate. From the StarDate, I make an RBN. I do the same for the ending date and time:

```
If EndDate and Endtime
  MyStarDate.MakeStarDate(EndDate,Endtime,EndStarDate)
  RBNEndDate = Self.RBNFromStarD(EndStarDate)
  ThisWindow.Reset(1)
End
```

If I already have the StarDates, I can start with the call to `RBNFromStarD`.

Once both RBNs have been constructed, I subtract and call the procedure to split it into its individual time components:

```
If RBNBeginDate and RBNEndDate
  GrossDifference = RBNEndDate - RBNBeginDate
  Self.TimeSplit(GrossDifference,DayComponent, |
    HourComponent,MinuteComponent,SecondsComponent)
  ThisWindow.Reset(1)
End
```

Why did I configure `TimeSplit` to take all of the return variables as variable parameters?

```
TimeSplit      PROCEDURE(*Decimal pTime, |
*Long pDays,              |
*Long pHours,             |
```

```
*Long pMinutes,           |
*Long pSeconds)
```

Look, I just conquered inheritance. Named groups, much less using them inside a class, are beyond my ... ambitions at this time

(By the way, the window `Reset` calls are so that you can see the computations as they occur. In real world apps, I wouldn't be using them.)

## Summary

Talking some time to think a problem through and understanding your data almost always lead to better code. In this case, not only is there less code, it is more readable and, therefore, more maintainable. Mostly, it is more reliable.

`StarET`, I think, needs to be deprecated from my time functions class. `RBNfromDT`, `RBNfromStarD` and `TimeSplit` need to be added.

However, I have not added the three new methods to my `StarDateClass`. Everything, for the moment, is still inside the .App. I found out something ... interesting that makes creating the class INC and CLW much easier. But, I've run out of time for this installment ....

Download the source

---

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

## Article comments

BACK TO TOP

# Tip of the Week: Two Ways To Compile

## By Dave Harms

Posted June 17 2011

I don't always compile apps using the Generate and Build option - sometimes I only want to compile the source to save time. For instance, if I've changed an export from an app, I only need to recompile the affected apps, I don't need to regenerate them.

Often I do this in the context of a solution that contains a number of applications. And there are two ways to do this: via the Solution Explorer, by right-clicking on a project node and choosing build, or via the App Pad. But I've noticed something odd. If I have circular references (an unfortunately reality sometimes), any build via the Solution Explorer returns an error about those circular references. Either this wasn't the case in C7, or I didn't notice it. In any case, the workaround is to compile the project via the App Pad. Just make sure you have the Build menu set to Build and not Generate and Build (Figure 1), unless of course you really do want to generate the app as well.

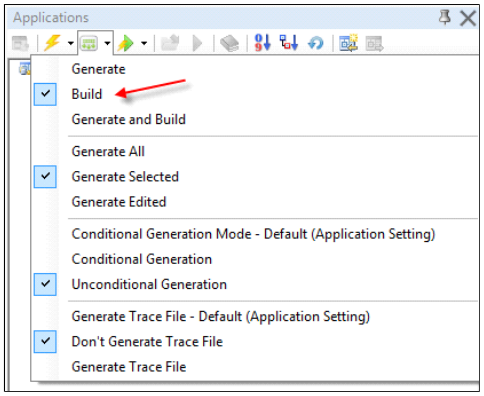


Figure 1. App Pad build settings

---

*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), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).*

## Article comments

BACK TO TOP

# ReplaceText

By Steven Parker

Posted June 23 2011

In "A String Class" I documented a class for manipulating strings. Constructing the class, I included a "replace text" function written by Peter Hermansen and posted on the news groups.

I didn't describe the `ReplaceText` method. I mentioned it to give credit to Peter for his contribution. Also, I knew that because it was in the class' prototype list, there would be questions if I hadn't mentioned it.

So I mentioned it. There were questions.

**The First Question**

Why doesn't `ReplaceText` blow up if you replace 1 character with 10 characters? Does Clarion keep expanding `pText`?

`pText` is a standard string, passed into `ReplaceText`. Therefore, its size is fixed. Therefore, the answer lays elsewhere, in the code.

The code for Peter's text replacement is:

```
ReplaceText  Procedure(STRING pOldStr, STRING pNewStr, STRING pText)
X    LONG,Auto
       Code
 Loop
   X = STRPOS(pText,pOldStr,True)
   If X
     pText = pText[1 : X-1] & Clip(pNewStr) & pText[X + LEN(Clip(pOldStr)) : LEN(Clip(pText)
   Else
     BREAK
   End
 End
 RETURN(pText)
```

`ReplaceText` replaces all occurrences of `pOldStr` in `pText` with `pNewStr`. The key to this code seems to be the `STRPOS` statement.

According to the LRM, `STRPOS` "Returns the starting position of a substring based on all parameters passed."

So, if `STRPOS` finds `pOldStr`, it returns the starting position of `pOldStr` in the target string. (This sounds a lot like how `INSTRING` works, except that I can use Regular Expressions in `STRPOS`' second parameter.)

So, for example, if I am searching for "fox" in "The quick brown fox jumped over the lazy dog," `STRPOS('fox', 'The quick brown fox jumped over the lazy dog', True)` will return 17. Position 17 in 'The quick brown fox..." is where "fox" begins.

If `STRPOS` returns a value, the magic begins. Peter's code parses the string into two portions, the portion up to, but not including the search string:

```
pText = pText[1 : X-1]
```

Translation: give me the string from the beginning to one space before the "found" position. He then adds in (concatenates) the new text:

```
& Clip(pNewStr)
```

Translation: add the replacement text. Finally, he computes and concatenates the remaining portion of the string:

```
& pText[X + LEN(Clip(pOldStr)) : LEN(Clip(pText))]
```

Translation: compute the length of the string being replaced, add that length to the starting position found by `STRPOS`. This is the position where the text being replaced ends. Then, using standard string slicing, get the part of the old string *after* the text to be replaced. When all three pieces are concatenated, if the result is longer than pText's length the excess is discarded by the runtime.

The whole operation looks like this:

```
pText = pText[1 : X-1] & Clip(pNewStr) & pText[X  + LEN(Clip(pOldStr)) : LEN(Clip(pText))]
```

Because this is in a Loop, `STRPOS` will find all occurrences of `pOldStr` in pText. Therefore, it does not seem logically different from calling:

```
X = Instring(pOldString, pText, 1, 1)
```

in a `Loop`.

**The Second Question**

What happens if the new string contains the old string? For example, replace "`100`" with "`10100`."

I hadn't thought about this when "requisitioning" Peter's code. But, the answer is "bad things," "very bad things."

Open the demo app (which you can download at the end of this article). On the main screen:

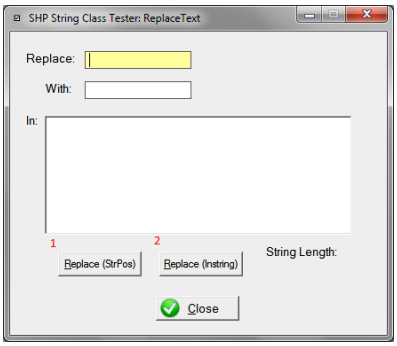


Figure 1: Demo app main screen

Type "fox" in the "Replace" field. Type "dog" in the "With" field. Type "The quick brown fox jumped over the lazy dog" in the "In" field. Then press the "Replace (StrPos)" button (button 1).

This should work fine.

Repeat the experiment and press the "Replace (Instring)" button (number 2). It, too, works.

Now, replace "fox" with "foxhound" in "The quick brown fox jumped over the lazy dog" and press the first button. The app will stop responding; it goes into an infinite loop and you will need to TaskManager it to death.

If

```
X = STRPOS('The quick brown fox ...', 'fox', true)
```

is indeed like

```
X = Instring(pOldString, pText, 1, 1)
```

(emphasis on the "1,1") then, in a Loop, "fox" will always be found at position 17 no matter how many times it is replaced by "foxhound." In other words, STRPOS always starts its search from the beginning of the string, just as I have speculated.

Now, repeat this second experiment. But press the second button.

That one works....

## INSTRINGing It

The difference between the two buttons is that the “Replace (Instring)” button employs a revised method that uses `INSTRING` instead of `STRPOS`.

`INSTRING` allows me to move the starting point of the search. So if

```
X = Instring(UPPER(Clip(pOldStr)),Upper(pText),1,1)
```

returns 17 into X, I can do a new search beginning after that point, thereby preventing checking the same block of text twice:

```
X = Instring(UPPER(Clip(pOldStr)),Upper(pText),1,18)
```

More generically, I can use a variable (`Y`) to specify the search-start position and increment it by the starting position to move the start point past the last “found point:”

```
Y = 1
L = Len(Clip(pOldStr))
Loop
   X = Instring(UPPER(Clip(pOldStr)),Upper(pText),1,Y)
   If X = 0
     Break
   Else
      pText = pText[1 : X-1] & Clip(pNewStr) & pText[X + LEN(Clip(pOldStr)) : LEN(Clip(pTex
      Y = X + L
   End
End
```

(I also computed and stored the length of the old string just once in this version of the code.) And that is why the “Replace (Instring)” button doesn’t go off into the æther replacing “fox” with “foxhound.”

## Summary

I’ve always maintained that the expression “there is no such thing as a stupid question” is nonsense. There is such a thing as a stupid question.

In fact, there is exactly one stupid question, the question that didn’t get asked.

So, I leave you with a question: Is `STRPOS` or `INSTRING` faster (especially for large strings)?

Download the source

---

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

## Article comments

*by Paul Blais on June 28 2011 (comment link)*

X = Instring(UPPER(Clip(pOldStr)),Upper(pText),1,Y) If X = 0 Break Else pText = pText[1 : X-1] & Clip(pNewStr) & pText[X + LEN(Clip(pOldStr)) : LEN(Clip(pText))]

If x = 1 then you "index out of range" String slice can be a nice optimized method for string handling but with replace you need to deal with the cases where your index will be zero.

You look at the string you are operating on as a head a middle and a tail. This is defined by the instring result as being the middle. We will overlook the overflow of the string with a replace that increases the length and overflows since you should have known better before you started:)

You have to deal with a replace that chops off the head or the tail so that the replace operation removes all the head or all the tail and generates a Zero value for the array index. You need to trap X = 1.

You can't use string slice to sllow all possible search and replace permutations.

Y = X + L

You need to make Y = X to allow the replace to be recursive. Consider a string with many multiple spaces and you want to remove all the double, triple, ... spaces with just one space. The function needs to be recursive and you replace all doubles with singles in a recursive pass.

---

*by Steven Parker on June 28 2011 (comment link)*

Paul, you're quite right. I did not think of finding the string at 1 and I did not think of replacing beyond the length of the target string.

Sounds like you need to write an article with a better ReplaceText method.

---

*by Steven Parker on June 28 2011 (comment link)*

But!

I just tested replacing the very first characters and it worked.

---

*by Paul Blais on June 29 2011 (comment link)*

Try deleting the first character as in replace something with null. It should blow up. Same for the tail end. That can't use a slice since the subscript goes to zero.

I also find the recursive option helpful but you might add a switch to use it so other replaces will be faster. Just use CHOOSE to test for the option and it's still one line of code. As in:

Y = X + CHOOSE(pRecursiveOption = TRUE, 0, L)

I wouldn't feel bad Steve, I went through all this with Capesoft in their StringTheory class. The early release missed all of this but has it all now. They did go farther and optimized slice operations where ever possible so I would keep all your code because it is all accurate. Trapping the 1 will allow you to use a SUB instead of a slice. Trapping the null replace value will also throw the head or tail away in those special situations. Once you do that you have a very very robust replace function that works with anything. I'm using it on one million byte strings reworking ASCII text documents and it does all I can throw at it. It can take a bit of processing but a lot faster than you might think even running a dozen or more passes with recursive replaces.

BACK TO TOP

# Tip of the Week: Multiple Installs

## By Dave Harms

Posted June 24 2011

There's often debate in the newsgroups about whether to install Clarion updates as new installs, or over existing installs. Many recommend a fresh install each time.

Like a few other Clarion devs, I take the approach that more is better, so I install each version of Clarion into its own directory.

That works because unlike Clarion through version 6, the Clarion bin directory in 7 and up is not on the path, at least not by default. This is an important difference, and it's one of the reasons why there's a "Copy referenced DLLs to output" option in the project settings. If the Clarion bin directory isn't on the path, then you'll need some other way for your EXE to find the runtime DLLs.

The good bit of not having the Clarion bin directory on the path is you don't have to jump through hoops to have different versions of Clarion installed at the same time. And if there's a big problem with the latest build, you can quickly go back to using the previous build (although if the APP or DCT format has changed you may have some extra work to do).

---

*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), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).*

## Article comments

BACK TO TOP

# Understanding Field History

By Steven Parker

Posted June 28 2011

How can I tell whether a specific field has changed?

How can I tell what the initial/previous value was in a specific field?

And, before I can answer either of these questions, how can I tell whether or not the record has changed?

## This is Clarion

This is Clarion, therefore this should not only be do-able, there should be multiple ways of doing it.

The easiest way is just to declare a local variable in the update form for each file field on the form. Later in `INIT` (any time after opening the file), "just" assign the current value of each field to its local "shadow." In `TakeCompleted, Before Parent call`, compare each local variable to the file variable. If any one of them is changed, I know both that the tested field has changed and that the record has changed.

Easy peasey and "surely, sir, you jest."

## ABC To The Rescue

For all but the *very* simplest files, declaring a local variable for each file variable (or, at least, for each variable in which I am interested), assigning initial values to them and, finally, checking them, one at a time, seems like an awful lot of work (not to mention some great opportunities for typos).

The ABC Libraries provide all kinds of methods for checking whether an update is needed and for actually performing the file update. Why not just use the built in functionality?

Taking my cue from Richard Taylor's original "Making the Transition to ABC" document, I started by searching for methods that looked like they might do what I need. There are three existing methods that look like good candidates:

- `ThisWindow.ChangeAction`
- `ThisWindow.SaveOnChangeAction`
- `ThisWindow.Update`

As it turns out, `SaveOnChangeAction` is not called during standard CRUD ops.

On record insert, `ChangeAction` is not called. (Why worry about inserts? See "Running Totals," new transactions have to update the running total.)

Both `ChangeAction` and `Update` *are* called during record change.

All this is well and good but it is *only* in Update that I can get both the previous and current values of a field. (I'll show you how I know this in a moment.)

To make a long story short, tracking changes this way is essentially as much work as the brute force approach of saving and checking variables in which I am interested. Furthermore, it entails using methods I don't normally (or, at least, don't often) use, compromising finding my code in the future.

How I know: in the demo app (instead of the ubiquitous People.App, this one is the School.App), see the `UpdateClasses` procedure. Run it; I have placed messages in all the methods mentioned above. I selected `CLA:ScheduledTime` as a field to track. If you make a change to `CLA:ScheduledTime`, you will see not only which methods are called and the order in which they are called but the "before and after" values in my "field of interest."

On the face of it, there doesn't seem to be a good reason why these methods appear to be so much work. Work with the UpdateClasses procedure in the demo and decide for yourself.

## EqualBuffer

Hang around the news groups, you will discover that *the* answer to "how can I tell if a record has changed?" is "`EqualBuffer`."

`EqualBuffer` is a `FileManager` method. The Language Reference states:

> The **EqualBuffer** method compares the managed file's record buffer, including any MEMOs or BLOBs, with the specified buffer and returns a value indicating whether the buffers are equal. A return value of one (1 or True) indicates the buffers are equal; a return value of zero (0 or False) indicates the buffers are not equal.

The Language Reference gives the following example:

```
IF ~SELF.Primary.Me.EqualBuffer(SELF.Saved) !check for any pending changes
  !handle cancel of pending changes
END
```

So,

```
If SELF.Primary.Me.EqualBuffer(SELF.Saved)
  ! record was not changed
ELSE
  ! record was changed
End
```

in `TakeCompleted, Before Parent call` should tell me whether the record was changed.

In the demo app, see the `UpdateCourses` procedure. `EqualBuffer` does correctly identify inserts and changes to existing records (not deletes, of course, see "Where Delete Occurs, Part 1").

However, if I need to know whether or not a specific field or fields were changed, I will require much deeper digging into the `FieldPairs` classes. While I am not in principle opposed to digging into core ABC classes, I don't think it's necessary.

In fact, I know it is not necessary to use `FieldPairs`. I know there are alternative because I've been checking records and fields for changes since I started using Clarion. And the important part is that the old way is easy as can be.

## The “Classic” Approach

The “classic” solution to the problem of knowing whether a record changes and, if so, whether a specific field of interest changed is to declare a local datum `LIKE(`*`pre`*`:Record)`. For example:

```
SavRecord   Like(STU:Record),PRE(SAV)
```

Then, after opening files, save the current record:

```
SavRecord = STU:Record
```

Finally, at the end of the procedure:

```
If SavRecord = STU:Record
  ! record was not changed
ELSE
  ! record was changed
End
```

tells me whether or not the record was changed.

(Note: this code works for both ABC and Legacy. Only the names of the embeds have been changed to protect the innocent.)

In the demo app, I am tracking `STU:Address2` in `UpdateStudents`.

Tracking a specific field is easy using my `SavRecord` variable. `SavRecord` is, effectively, a `Group` structured exactly like the underlying file's `Record` declaration. Therefore,

```
If Sav:Address2 = STU:Address2
  Message('Address2 unchanged','Nope')
ELSE
  Message('New Address2 is ' & STU:Address2 & |
    '<13,10>Old Address2 was ' & SavRecord.Address2,'Yup')
End
```

If you're a fan of dot syntax,

```
If SavRecord.Address2 = STU:Address2
```

also works.

Because `SavRecord` contains the original value(s), I have the original value of any variable in the record. Because the update form has the current value, I always know the previous and current values and can act on them if I need to.

## The “ABC Way”

For as long as I can remember, Clarion for Windows’ template-based forms, by default, create a local

variable similar to my `SavRecord` variable. In my one remaining Legacy app, for example, my `UpdateChecks` procedure declares:

```
SAV::CHE:Record       LIKE(CHE:Record)
```

If you trace this "`SAV::`" variable in a Legacy Form, it is used to trigger file updating, pretty much as described above.

In ABC apps, a similar variable is declared:

```
CurrentTab           STRING(80)
DOW                  STRING(10)
ActionMessage        CSTRING(40)
! [Priority 3000]

EnhancedFocusManager EnhancedFocusClassType
! [Priority 3580]

History::ART:Record  LIKE(ART:RECORD),THREAD
! [Priority 4800]

! Window Structure
QuickWindow          WINDOW('Articles Info'),AT(,,35
  MDI,HLP('UpdateArticles'),MSG('Article Information
```

Figure 1: "SavRecord" variable declaration in ABC

Thereafter, `History::STU:Record` is only used in other ABC methods. Figure 2 shows how a file and its fields are added to the `WindowManager`'s "history".

```
! Procedure setup standard formulas
SELF.HistoryKey = CtrlH
SELF.AddHistoryFile(ART:Record,History::ART:Record)
SELF.AddHistoryField(?ART:ArticleID,1)
SELF.AddHistoryField(?ART:SiteID,2)
SELF.AddHistoryField(?ART:Title,4)
SELF.AddHistoryField(?ART:Description,5)
SELF.AddHistoryField(?ART:URL,6)
SELF.AddHistoryField(?ART:DatePosted,3)
SELF.AddUpdateFile(Access:Articles)
SELF.AddItem(?Cancel,RequestCancelled)
! [Priority 7300]
```

Figure 2: Usage of history variable

The Help says this about `AddHistoryFile`:

> The **AddHistoryFile** method adds a history file to the WindowManager object. AddHistoryFile sets the file's record buffer and a corresponding save buffer so the WindowManager can restore from the save buffer when the end user invokes the history key (or FrameBrowseControl ditto button).

The `AddHistoryField` method pairs file fields with their window controls.

So, there does not appear to be much to help in my current quest to know what data has changed. But reading a bit further in the `AddHistoryFile` documentation reveals another method:

## SaveHistory

From the help:

> The SaveHistory method saves a copy of the fields named by the AddHistoryField method for later restoration by the RestoreField method.

implies that calling this method will save the record buffer. Therefore, if called before any edits can be made, History::STU:Record (or whatever your file prefix is) will be a copy of the original record.

> The AddHistoryFile method names the file and record buffers from which fields are saved and restored. The AddHistoryField method associates specific fields from the history file with their corresponding WINDOW controls. The SaveHistory method saves a copy of the history fields

So, late in `INIT`:

```
Self.SaveHistory()
```

ought to save the current record buffer.

To determine if the record has been changed, in `TakeCompleted, Before Parent call`:

```
If History::pre:RECORD.pre:label <> pre:label
  ! record has changed
Else
  ! record has not changed
End
```

In the demo app, I track `TEA:FirstName` in `UpdateTeachers` with:

```
If History::TEA:Record = TEA:RECORD
  Message('No change in teacher record','Teacher Not Changed')
ELSE
  Message('Teacher changed <13,10>Previous First Name ' &|
    History::TEA:Record.TEA:FirstName &'<13,10>New first ' &|
    'name ' & TEA:FirstName,'Teacher Updated')
End
```

Note the line in bold face. That's how I can check any of the file fields, similar to comparing Sav:`Address2` to `STU:Address2` in the code above.

## On Reflection

These two approaches differ only in minor details. They are logically identical. In one, I declare the variable; in the other, it is declared for me. In both, I have to save the initial state of the record, either by invoking a method or by making an assignment; this makes little difference.

The big difference is in the comparisons at the end:

```
If SavRecord = STU:Record
```

versus:

```
If History::TEA:Record = TEA:RECORD
```

and

```
If Sav:Address2 = STU:Address2
```

versus:

```
If History::TEA:Record.TEA:FirstName = TEA:FirstName
```

And I do see a significant difference between both pairs. The “classic” method is quite a bit less typing.

## Hybrids, They're All The Rage These Days

If Clarion provides a variable, why not just use it instead of declaring my own? Just make the assignment;

```
History::ENR:Record = ENR:Record
```

And be done with it. Then, use it, as in “the ABC way” in `TakeCompleted`:

```
!If SavRecord = ENR:Record
If History::ENR:Record = ENR:Record
  Message('Record not changed')
ELSE
  Message('This record was changed')
  !Message('Original mid term grade ' & SAV:MidtermExam &  |
  ! '&13,10>New grade ' & ENR:MidtermExam)
  Message('Original mid term grade ' &  |
   History::ENR:Record.MidtermExam & '<13,10>New grade ' & |
   ENR:MidtermExam)
End
```

In the demo app, I track `ENR:MidtermExam` (grade) in `UpdateEnrollment`. Obviously, it works as expected (with the extra typing in `TakeCompleted` noted earlier).

I don’t see that it offers anything just doing it the old fashion way doesn’t but I had to look at it.

## Global Code vs. Local Code

Must I always add code to my update procedures? The Dictionary Editor, after all, offers embedded triggers. Using a trigger, I can put my code into the dictionary and be done with it.

However, a trigger in the dictionary is not possible. There is no place available in the dictionary to store the initial state of the record:

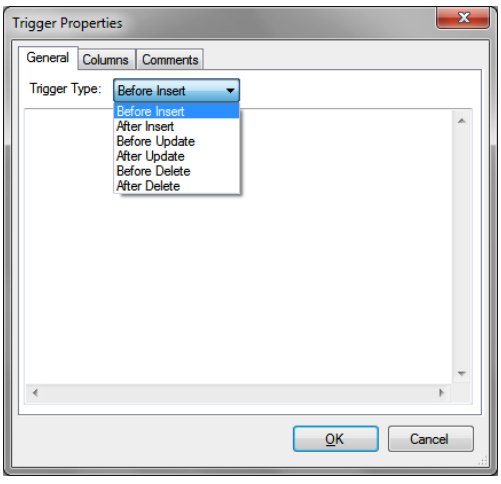


Figure 3: Available Trigger Points

There are trigger points available for before and after CRUD ops. But, there isn't a trigger point available to save the original record buffer. So, to use a dictionary trigger still seems to require hand coded embeds.

However, in "Where ... Stuff Occurs: Deriving the FileManager" and in "Deriving The FileManager In A Multi-DLL App," I showed that there are a number of global embeds related to CRUD operations. Indeed, there are all manner of global `FileManager` embeds surrounding virtually every step of every `Insert`, `Update` or `Delete` operation (plus many other things).

Rick Martin outlines the steps to save the current record and check a specific field, on a record change, using the `FileManager` global embeds:

> Go to your data dll, global embeds.
>
> In the Global Objects find the File Manager for your table.
>
> In the `Init` method near the end embed this code:
>
> ```
> Self.SavePreviousBuffer  = True
> ```
>
> Now in your PostUpdate embed you can use `Buffer.FieldName` to test the old value. For example:
>
> ```
> If  Buffer.LastName = CUS:LastName
>   Do something
> ```

This has the virtue of allowing different actions after an insert, update or delete. It also has the virtue of making post-Delete operations a bit easier to control. It has the shortcoming that it is economical *only* if there are multiple update forms for a given file and, if there are, "Do something" ceases to be optional. Once coded into a `FileManager` embed, "Do something" will *always* be done.

If there is good reason to create multiple update forms for one file/table and if a specific CRUD op must always do specific post-processing, as they say, there is no substitute.

## Summary

I've shown five ways of determining whether or not a record has changed (six if you include the brute force method first mentioned; seven if you include deriving the FileManager). Three of them also give me access to the before and after values in any file variable I may want to know about. (I was right, this *is* Clarion.)

None of them is very difficult nor anything fancy. And all are entirely accurate.

Download the source

---

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

## Article comments

*by Paul Blais on June 29 2011 (comment link)*

The DCT trigger "Before Update" does not fire if the History record has not changed. ABC has already done the check for you. If you add a Return Level:Notify the update can be cancelled from the trigger and similarly for delete or insert.

BACK TO TOP

# GainFocus: When and How

By Steven Parker

Posted June 29 2011

`Event:GainFocus` was the subject of a recent thread on one of the news groups.

I would expect that Event:GainFocus was triggered any time I selected/clicked on a window, after having selected/clicked on another window. If I open a browse on customers, I would expect GainFocus to fire. If I then click on a browse on orders (assume it was already open), I would expect GainFocus to fire. And, when I re-select the customer browse, I would, yet again, expect `GainFocus` to fire.

Similarly, if I click on the desktop or another app, on returning to the app I started with ... well, you get the idea.

However, the protagonists in the news group thread reported different experiences. One reported only getting the event when clicking on another app and returning. Others reported other behaviors entirely out of keeping with the documentation:

> The window is gaining input focus from another thread. This is the event on which you restore any data you saved in `EVENT:LoseFocus`. The system is modal during this event.
>
> `EVENT:GainFocus` is not generated until `EVENT:LoseFocus` is processed (if focus was on another window of the same program).

So, what's the problem? The problem is that the protagonists of the thread are very competent Clarion developers, one of whom I count as one of my early Clarion teachers. If these folk are having issues with something that, on its face, is so simple, `GainFocus` needs some investigation.

## Testing Strategy

Sophisticate that I am, I started with the infamous People.APP. People features one browse, one form and a couple of reports. One browse isn't enough for testing so I added a second browse-form pair.

In each browse and form, I embedded a `Message()` in `Event:GainFocus`:

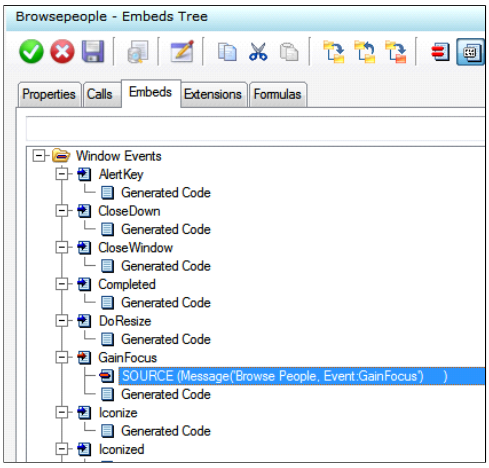


Figure 1: Event:GainFocus embed

The `BrowsePeople` browse procedure uses Edit-in-Place. This gives me three window procedures (`BrowsePeople`, `BrowseStates` and `UpdateStates`) to switch between, seeing if my `Message()` appears.

When I open `BrowsePeople`, I get my message:

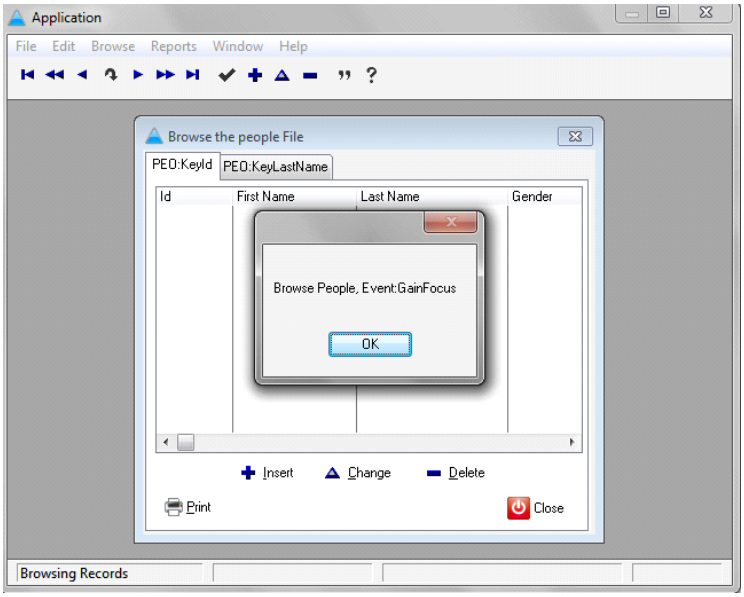


Figure 2: Opening BrowsePeople

Because `BrowsePeople` uses EIP, the `Message()` doesn't appear if I try to update the `People` file.

When I open the `BrowseStates` browse, I also get my message:

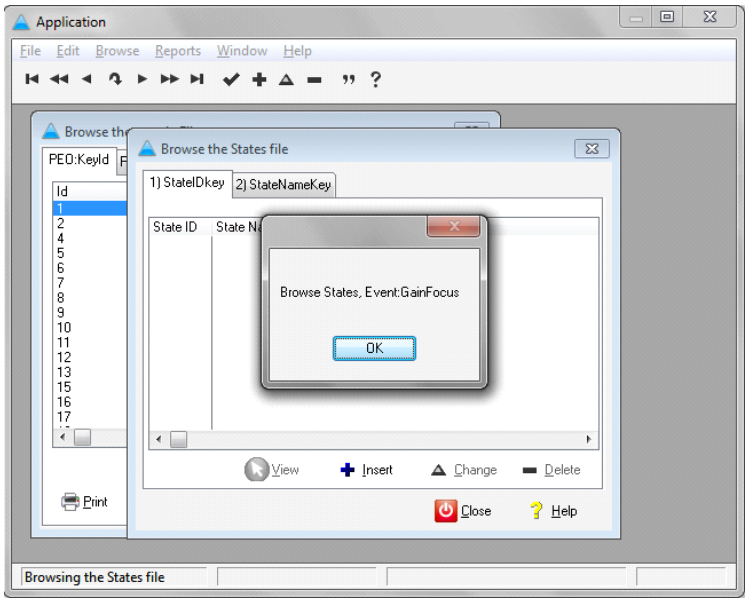


Figure 3: Opening BrowseStates

And, with both browses open, if I click on `BrowsePeople`,

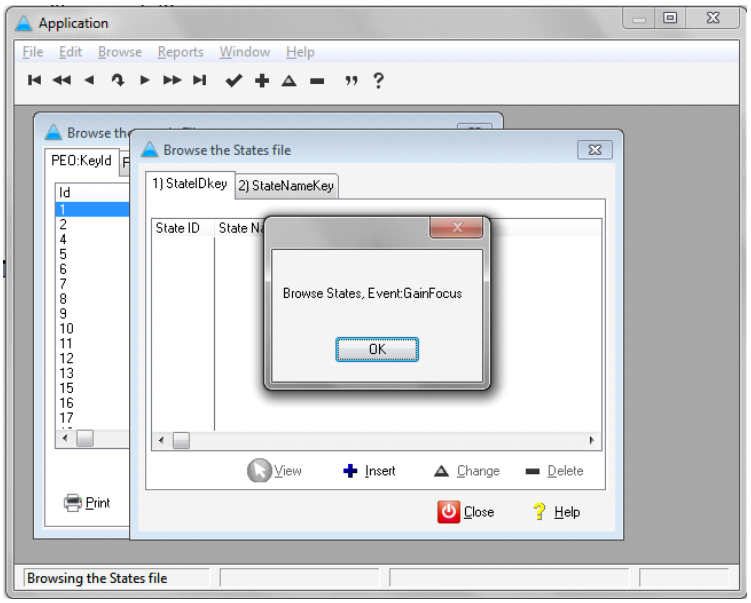


Figure 4: Switching back to BrowsePeople

I also get the `Message()`.

When I click on another app (or the desktop) and click back, I get the message. When I Alt-Tab to another app and Alt-Tab back, I get the `Message()`. (Open the demo app and try this for yourself – the demo was created with C8.0.8274.)

## Does Having a Frame Make a Difference?

The demo app, People.EXE, has a frame. Does an app without a frame make a difference?

To test this, I wizarded up a browse on People.TPS and made it the main procedure of an app

(NoFrameTest.APP).

Start up NoFrameTest.EXE and my `Message()` appears. Click or Alt-Tab to another app and return to NoFrameTest.EXE and my `Message()` appears.

It does not seem that a frame makes a difference.

## Summary

Well, `Event:GainFocus` may have been unreliable in the past. But, as of 8274, it seems to be working pretty much as expected. Creating an app to test it wasn't all that hard either.

Download the source

---

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

## Article comments

BACK TO TOP

# Tip of the Week: Multiple Installs for Team Development

By Dave Harms

Posted June 30 2011

This week's tip is a follow up to last week's tip on installing multiple versions of Clarion. In some shops, team development means everyone works on one server, via Remote Desktop Connection, and all share the same Clarion install (with one serial number for each user, of course). This actually works quite well for geographically distributed teams. But upgrading Clarion can be a little more complicated.

In this situation I've adopted a modified version of the each-version-of-Clarion-in-its-own-directory approach I described last week.

The very first install goes into the default directory, e.g.

```
C:\Program Files (x86)\SoftVelocity\Clarion8
```

I'll make a copy of this directory and name it after the build number, e.g.

```
C:\Program Files (x86)\SoftVelocity\Clarion8.0.8399.
```

## Installing new versions

Whenever I get a new version of Clarion, I install it into a directory which I call Clarion8.install:

```
C:\Program Files (x86)\SoftVelocity\Clarion8.install
```

I then make a *copy* of this directory and name it after the build number:

```
C:\Program Files (x86)\SoftVelocity\Clarion8.0.<buildnumber>
```

For example:

```
C:\Program Files (x86)\SoftVelocity\Clarion8.0.8461.
```

I then delete the Clarion8.install directory.

The reason I make a copy of the directory, rather than just naming Clarion8.install after the build number, is I want to invalidate any shortcuts created by the installer. If I rename the directory then the shortcuts will get updated. I don't want those shortcuts to work at all, in the unlikely event that someone tries to use one. I want everyone using shortcuts that point to the original Clarion8 install. And because I'm always copying updates into my Clarion8 directory, rather than renaming directories, the original Clarion8 shortcuts will always work.

If I started with 8399, and moved to 8461, I'll now have the following directories:

```
C:\Program Files (x86)\SoftVelocity\Clarion8
C:\Program Files (x86)\SoftVelocity\Clarion8.0.8399
C:\Program Files (x86)\SoftVelocity\Clarion8.0.8461
```

Everyone will be running out of Clarion8, and initially that will be build 8399. When it comes time to upgrade, I just make sure everyone's out of Clarion and then I

1. Sync Clarion8 to Clarion8.0.8399 to make sure my 8399 backup is good, and
2. Sync Clarion8.0.8461 to Clarion8

Because I have each version of Clarion in its own directory, I can test versions before releasing them to the team, and I don't disrupt the team while I'm making any adjustments to the install, such as integration template customizations.

In fact, this team uses a lot of customized templates, so I have two directories for each version of Clarion. One is the stock install, and the other is the customized install. I sync to our working Clarion 8 install from the customized installs, but I keep the stock installs around for reference against the customized install, and so I can easily compare one stock install to another to see if there were template changes that need to be migrated into our customized version of the templates.

It's definitely a bit of work keeping all those installs of Clarion around, but it's far less disruptive to the team to do the actual installations in isolation *and* it gives us an easy path back to previous versions (app and dct format changes notwithstanding).

---

*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), and has written or co-written several Java books. David is a member of the American Society of Journalists and Authors (ASJA).*

## Article comments

BACK TO TOP

# Testing Classes, Creating Classes

## By Steven Parker

Posted June 30 2011

When last sighted, `shpStarClass` needed to have one method deprecated and three new methods added.

Deprecating a method is not that hard. Delete the prototype from the INC. Then delete the method definition from the CLW. As long as I don't delete too much (my mouse skills *are* deprecating after all) there are no significant intellectual challenges here.

Adding new methods to class files is (mechanically) just the same as creating them in the first place. And this is where I consistently make mistakes. I can type a prototype into an INC file well enough, usually. Then I usually copy and paste the procedure declaration line into the CLW and type (or copy and paste) the code into the CLW.

The trouble I have in the CLW is that I sometimes forget to remove (or comment out) the return data type on the first line of the code block (when there is a return value). For example:

```
TimeUpTime    Procedure(Real pTimeAllowed),Real
```

which, in the CLW, should be:

```
TimeUpTime    Procedure(Real pTimeAllowed) !,Real
```

I also tend to forget to add the class' label to the method definition:

```
TimeUpTime     Procedure(Real pTimeAllowed)
X   LONG
TMP Real(15.8)
Day Long

    CODE
     ! ...
```

which should be:

```
shpStarClass.TimeUpTime     Procedure(Real pTimeAllowed)
X   LONG
TMP Real(15.8)
Day Long

    CODE
```

```
! ...
```

Finally, if there are calls to other class methods from the current method, and I did my initial testing outside the class CLW, the code will contain references to an object. Of course, that object no longer exists. I tend to forget to change these calls to "`Self.`"

All of my little problems raise the question, in my mind at least, of how my methods are originally created and tested.

## Methods of Madness

I can think of only a very few ways to create classes, specifically the INC and CLW files. In my mind, these are closely bound to how I test them. To me, creating and testing any code are inextricably bound together.

One way, of course, is to create two new text files and start typing. That's still a bit ambitious for me.

Because I cannot conceive of any kind of code without thinking about how I will prove the code works correctly – any code – testing is uppermost in my mind. That's just me, I guess.

When I create a new procedure, I start with some idea – rarely a written spec and I have never had a complete description – of what the procedure is supposed to do. I have an idea of the business need to be addressed. My testing follows from my understanding of the business need, as described to me.

Modifying an existing procedure is, theoretically, not that much different from creating a new one. Theoretically, and let's leave it at that.

But, a new class or a change to an existing class doesn't have the context of an app. By definition, a class is (or should be) "application-agnostic." A class neither knows about nor depends upon nor "cares" about the app in which it is called.

Dave Harms has written at some length about his method of unit testing. In his paradigm, the test procedure that exercises the class is incorporated in a DLL APP. That DLL is loaded by a unit testing application, the test procedure is called, and the application reports on the results returned by the test procedure. (*If* I understand what he has been saying.)

For me, a much less class-sophisticate, I design a new app around the things I expect the class methods to do. Each of my last several articles has been accompanied by one (or more) such "tester apps." To keep as close to my comfort zone as possible, I create each of the procedures intended to go into the class as a separate procedure within the tester app. In this way, the testing window calls the procedures in substantially the same way as they will be called when "promoted" to the final class.

More importantly, everything I know about testing and debugging can be brought to bear, as is. That's the important part, "as is." I don't have to adapt my existing work style to anything new. Or, at least, the adaptation is minimal.

These tester apps incorporate a call to each candidate procedure in as simple and straight forward a manner as possible. Even if a method is not intended for "public use," I create a visible test. This allows me to see that each method does what I expect it to do. And, just as in previous articles, I construct a set of test cases.

Once functioning as expected, I simply copy and paste the procedure's Data and Code Sections from the .APP file into the CLW (or, I try to).

And, that's where things go missing, like the class Label on the method declaration.

## Eureka!

In creating the testers for my `StarDate` class, instead of creating procedures – I *think* that this is what I had done in the past – I created local methods within the local `WindowManager` class (sometime, erroneously, referred to as "locally derived methods"):

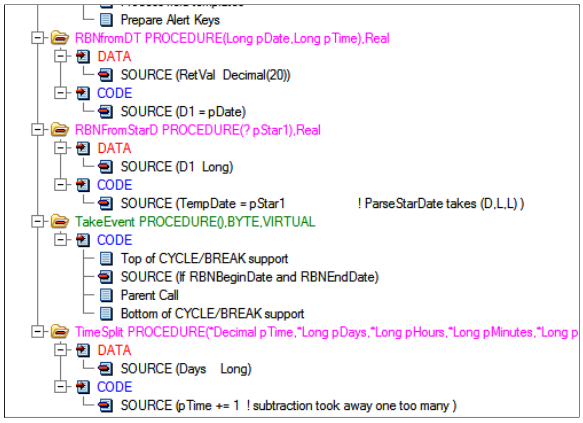


Figure 1: Testing via locally derived methods

I reasoned that since the code was intended to go into a method, why not start it out as "in a method."

Why not indeed?

What I saw, when I viewed the code from "Source view" looked like this:

```
ThisWindow.RBNFromStarD PROCEDURE(? pStar1)

! Start of "New Class Method Data Section"
! [Priority 5000]
D1  Long
T1  Long
R1  Decimal(20)
TempDate DECIMAL(15,8)
D2  Decimal(20)
T2  Decimal(20)

! End of "New Class Method Data Section"

  CODE
  ! Start of "New Class Method Code Section"
  ! [Priority 4000]
  TempDate = pStar1                    ! ParseStarDate takes (D,L,L)
  !Self.ParseStarDate(TempDate,D1,T1)    ! but RBN needs D's
  MyStarDate.ParseStarDate(TempDate,D1,T1)
  T1 += 1
  D2 = D1                              ! possible to use L's but assign to D?
  T2 = T1                              ! actually, this code is passing L's into a
  R1 = Self.RBNfromDT(D1,T1)
  Return R1
```

Figure 2: Source view of my method code

Other than the template comments, in green, and the "`ThisWindow`" prefix, it looks exactly like what I needed in the final CLW (and, if, in the past, I developed new methods as locally derived methods, I just didn't see this before).

It struck me that I could "just" copy the whole of the code here and paste it into my CLW. Since I am focused on the cut and paste, I immediately started by removing the template comments. Thus, because I am in "edit mode," I did not forget to change "`ThisWindow`" to "`shpStarClass`." Neither did I

forget to change "`MyStarDate`" to "`Self`."

Once I think I have correct class files, I edit my .APP to include and instantiate the class. I change the calls from "`Self`" to "`MyStarDate`" (or however I've instantiated the class) and run a final test.

And I didn't get any errors, as I usually do!

To ensure that I don't accidentally call one of my locally derived methods in final testing, I actually go into the New Class Methods prompt and delete the locally derived methods. If you check the demo app, you will see that the three new methods created last time are still in the embed tree. But, they are now orphans. (The final class files are also included with the demo app.)

## If only ...

If only I could copy and paste the prototypes that need to go into the INC file. It's not that creating the prototype list is difficult but why risk a typo if I don't have to? Of course, the generated code for the ThisWindow object – when the new methods were "still" locally derived methods -- contains exactly what I need for a copy and paste:

```
! End of "Data for the procedure"
ThisWindow           CLASS(WindowManager)
Ask                    PROCEDURE(),DERIVED
ChangeAction           PROCEDURE(),BYTE,DERIVED
Open                   PROCEDURE(),DERIVED
PrimeUpdate            PROCEDURE(),BYTE,PROC,DERIVED
Reset                  PROCEDURE(BYTE Force=0),DERIVED
Run                    PROCEDURE(),BYTE,PROC,DERIVED
SetAlerts              PROCEDURE(),DERIVED
TakeAccepted           PROCEDURE(),BYTE,PROC,DERIVED
TakeDisableButton      PROCEDURE(SIGNED Control,BYTE MakeDisable),DERIVED
TakeEvent              PROCEDURE(),BYTE,PROC,DERIVED
Update                 PROCEDURE(),DERIVED
RBNfromDT              PROCEDURE(Long pDate,Long pTime),Real ! New method added to
RBNFromStarD           PROCEDURE(? pStar1),Real              ! New method added to th
TimeSplit              PROCEDURE(*Decimal pTime,*Long pDays,*Long pHours,*Long pMin
                     END
```

Figure 3: My prototypes *in situ*

Again, all I need to do, after pasting, is remove the "green slime."

## Summary

There it is ... once I have decided that I need a class and that I am going to create one, I really don't have to wander very far from my normal working methods, i.e., my comfort zone.

I need to design an app to test the procedures ... uh, methods, I intend to create and include in my class. Each method-to-be is created as a locally derived method. Tests are run, code is debugged. When all is ready ... copy, paste and off to the races.

Download the source

---

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

## Article comments

*by Russell Eggen on July 1 2011 (comment link)*

Steve,

That definitely fits within the LPS "rules" .

Russ

BACK TO TOP