Clarion Magazine

Reborn Free CLARION online

Home **COL Archives**

Date Checking, Time Stamping

There are a number of reasons why you might want your application to check and act on dates and times. Perhaps you want to time your software out after a certain number of days. If your software is a demo or leaseware, the ability to do this is critical. Similarly, if you want to run automatic end of day processing or check for fiscal year rollover, you need to be able to verify dates. Steve Parker shows how it's done.

Posted Monday, January 07, 2002

A Template For Copying Fields Between Files

If you're like most Clarion developers, you frequently have to copy fields between files. Clarion's deep assignment is perfect for this, right? Wrong, says Andrew Guidroz.

Posted Thursday, January 10, 2002

Handing COM Events - Part 1

The one area of COM that the Clarion OLE control is particularly weak in is receiving events. When the OLE control works, it works very well and is very simple to use. However, more often than not, it does not work, and in a case like that you'll need Jim Kane's OLE event code. Read on. Part 1 of 2.

Posted Friday, January 11, 2002

November 2001 PDF Now Available

The November 2001 PDF is now available for download.

Posted Friday, January 11, 2002

The ClarionMag Sweepstakes January Draw Is Coming Soon!

We've revised the draw dates for the Clarion Magazine Sweepstakes! There will be an interim draw on January 31, for subscriptions and product prizes; if you win on the interim draw, you're still eligible for the final draw in

Enter the ClarionMag Sweeps!

Subscribe, **Renew**, or **Refer a Friend** and you could win a **Compaq iPAQ** or an **ETC-III registration!**

There were 11 winners in the January 31 interim draw.

No purchase required.

News

INN Bio: Mihai Palade

Advertise On The Clarion Connection

Digital Alarm Clock Demo App

New Icetips Previewer Demo

New Home For Schoeffler & Rau Datensysteme

SAPDB Available For Download

Icetips Previewer Released

SealSoft xXXL Pack

February. Enter by renewing or subscribing, or by referring new subscribers. First prize is your choice of a Compaq iPAQ or an ETC 2002 conference registration. There are other great prizes too!

Posted Monday, January 14, 2002

Weekly PDF for January 6-12, 2002

All ClarionMag articles for January 6-12, 2002 in PDF format.

Posted Monday, January 14, 2002

Handing COM Events - Part 2

The one area of COM that the Clarion OLE control is particularly weak in is receiving events. When the OLE control works, it works very well and is very simple to use. However, more often than not, it does not work, and in a case like that you'll need Jim Kane's OLE event code. Read on. Part 2 of 2.

Posted Wednesday, January 16, 2002

Write A Word Processor In Five Minutes

Inspired by a 30 page tutorial on creating a text editor in Delphi, Vince Du Beau decided to see how he could improve on the example in Clarion. The result: a word processor in five minutes.

Posted Thursday, January 17, 2002

Bob Zaunere To Keynote ETC-III

Bob Zaunere has been confirmed as the keynote speaker at ETC-III, May 21-24 2002 in Gatlinburg, Tennessee. Bob Foreman will be presenting on ClarioNET. NOTE: Conference space is limited, and ETC organizers report registrations are coming in faster than for either of the previous two ETC events.

Posted Friday, January 18, 2002

Interfacing With C++ Part 1

This article demonstrates how Clarion interfaces and C++ abstract base classes can be freely interchanged between Clarion, TopSpeed C++ and Microsoft Visual C++. Not only does this provide a convenient conduit for mixed language development, it also allows objects to be shared across languages. In practice it is possible to build a very powerful

DateTime Clock Special Pricing Ends Thursday

CPCS For C55 Now supports Internet Connect Under ABC

CPCS V5.50d Previewer Supports Prompt, Tip, And Icon Customization

New Icetips Previewer Demo

New Wizard Template And Classes Released

DOS Printer v5 Released

AnalyZe.IT

Dennis Evans' Web Site

Clarion TWAIN Demo And Template

SealSoft xWinSet v2.09 Released

Clarion Handy Tools Thin Client Data Demo

SysDTP Date And Time Picker

xNotes v1.2 Released

Dennis Evans Featured INN Bio

Gitano G Build Support

Free Data Dictionary For A Software Project Management System

Shareware Version of EmailData Template

C++ library by simply wrapping existing or third party code within a Clarion compatible interface. Part 1 of 2.

Posted Friday, January 18, 2002

Interfacing With C++ Part 2

In this second of two parts, Gordon Smith concludes his demonstration of how Clarion interfaces and C++ abstract base classes can be freely interchanged between Clarion, TopSpeed C++ and Microsoft Visual C++.

Posted Monday, January 21, 2002

Weekly PDF for January 13-19, 2002

All ClarionMag articles for January 13-19, 2002 in PDF format.

Posted Wednesday, January 23, 2002

First Field, Required Field

When you cancel a form that has field validation code, does that code still execute? If so, you have the "required field lookup blues." Here's Dr. Parker's prescription.

Posted Friday, January 25, 2002

Weekly PDF for January 20-26, 2002

All ClarionMag articles for January 20-26, 2002 in PDF format.

Posted Monday, January 28, 2002

Getting A Handle On The System Tray

One main benefit of Windows programming is being able to have multiple applications open at once. The problem with this is that if you have ten programs active at once, you have ten programs cluttering up your task bar, and ten icons to Alt-Tab through to land on your desired application. So - the system tray to the rescue! This article by James Cooke (not Gordon Smith, as earlier indicated - my apologies James! ed.) covers the basic steps required for parking an app in the system tray, and responding to events on that icon.

Posted Wednesday, January 30, 2002

ABC/Legacy Available

CPCS Preview Now Handles Varying Sized Pages

Freeware Capitalize Template Updated

RAS (Dial-Up) Library Freely Available

xWord Library v1.6 Released

Handy Tools Example Apps Revised

EmailReport ABC Version 1.1

Free EFT/Credit Card Template Beta

New Clarion Site

EnhancedScrollClass C3PA Compliant And C5.507 Compatible

CPCS Adds Report Concatenation Feature

New TeamIDD Pricing Structure

Gitano Software Support Forum

Upgrade To A Gitano Bundle Promo

DOS Printer version 4 Released

xQuickFilter v2.09 Released

Sterling Data New Year Bundle Discounts End Jan 10

Juan Domingo Herrera Featured In INN Bio

Compile Manager 2 Available

Translator Plus Legacy Template Released

TeamIDD Distributed Source Control

Shareware EmailReport ABC

RInstall Update

Fomin Report Builder Public Support Forum

PSI's TimeTrak 1.1 Released

INN Bio For Jan 3, 2002

New Scripting Language For Clarion And Business Users

ForeHelp Closes Doors

Clarion Magazine

Home COL Archives

Topics > Tips/Techniques > Dates and Times

Date Checking, Time Stamping

by Steven Parker

Published 2002-01-07

There are a number of reasons why you might want your application to check and act on dates and times. Perhaps you want to time your software out after a certain number of days. If your software is a demo or leaseware, the ability to do this is critical. Similarly, if you want to run automatic end of day processing or check for fiscal year rollover, you need to be able to verify dates. Or, if you need to archive or to purge archives ... well, you get the idea.

The conventional wisdom is that you cannot date-lock a Clarion program. You cannot do so, says the conventional wisdom, because you are forced to use the system clock and the system clock cannot be assumed to be correct. If you can't rely on the clock, then it also follows that you cannot reliably time-stamp either.

It is obvious that you cannot, in fact, rely on the system clock. Users can and do change dates and times (I do so all the time when testing my applications). The batteries that drive the clock can go bad. Some PCs simply lose time (I have a fairly new notebook that reliably loses several minutes a week).

All of this has been obvious since I first addressed this problem in "Time Stamping With Confidence" (Clarion for Windows Magazine, 1, 6, March/April 1997). Equally obvious, to me at least, is that the conventional wisdom simply misses the point. That's why I wrote that article in the first place.

The point is not to ensure that each end user's PC clock is dead accurate. The point is that dates and times simply need to be *reliable*. That is, dates cannot be allowed to roll back i.e., once a date is set, no earlier date-value can be accepted. The same is true of times. If any given date-time measurement is later than any previous measurement, that is good enough for reliable time stamping and, therefore, date locking. The date and/or time may not be exactly correct but they are, at least, in the correct order.

Phrased this way – and I am a long time believer in the fact that the answer to a question often, if not usually, hinges on how the question is phrased – date locking or date limiting your software is really no more difficult that limiting the number of records in a file. I'll explain that in a moment.

Once I reformulated the question and redefined the conditions, I found that implementing date locking, time stamping, whatever you want to call it, is entirely a matter of determining what needs to be checked, what order to do the checks and what to do about the results. In other words, it's *all* in the planning; the execution is straightforward.

What I need to know

To add date/time stamping/locking to an application, you'll need a few fields in an encrypted configuration file. While encryption can be broken, anyone willing to go to the lengths required is going to break most protection schemes, so I don't think it worth worrying overly much about these types of "customers." Encrypt the file. The idea is not to make date/time locks unbreakable, that can't be done (there's always someone out there more clever *and* more devious than you are), but sufficiently inconvenient that most folks won't bother.

Field	**Description**
`Date first used`	This is needed to calculate the timeout date.
`Date last used`	This is needed to check that the current start up of program is at least as late as the previous use.
`Time last used`	Ditto.

`Number of days allowed`	If an action is to take place after a pre-set number of days, this datum is necessary.

These define a file structure like this:

```
CONFIG FILE,PRE(CFG),DRIVER('TOPSPEED'),RECLAIM,OWNER('♥')
RECORD                RECORD
FirstDate               LONG
LastDate                LONG
DaysAllowed             SHORT
LastTime                LONG
                  . .
```

Acting after X days

Timing out a demo or taking action after a certain number of days is quite easy with this configuration file.

Late in `INIT`:

```
Access:Config.Open
Set(Config)
Next(Config)
If Today() > CFG:FirstDate + CFG:DaysAllowed
  Message('Please pay for this program.',   &|
          'Evaluation Period Over',ICON:Hand)
  Access:Config.Close
  Return Level:Fatal
End
```

You may notice that I do not check for errors in this code. There are two reasons for this. First, in the app from which this is taken, the file was opened and checked for errors a bit earlier. Second, I distribute a copy of the configuration file as part of the installation and the file cannot be created by the app. If there is an error on opening or reading the configuration file, something is seriously wrong and the user should be dumped out of the program. If you decide to use this technique, "salt to taste."

If you need to take some specified action, just substitute a procedure call and appropriate disposition for `Access:Config.Close`.

The real problem is ensuring that `Today() > CFG:FirstDate + CFG:DaysAllowed` is reliable, that no more than the requisite number of days have passed and that the user hasn't rolled the system date back to try and beat you.

The strategy I developed recognizes that system clocks can be wrong, for whatever reasons. However, because *I* am in control of what goes into or out of the configuration file, *I* can enforce a certain degree of order.

I cannot guarantee that the date and time are correct. But I **can** ensure that dates and times go in the generally approved direction: forward only. I can also minimize the user's ability to lie to me.

First program use

The basic test is:

```
If Today() > CFG:LastDate
```

However, it makes no sense to test this until I have a value in `CFG:FirstDate`. `FirstDate`, of course, should be initialized the first time the program is started.

But the first time the program is started, I have nothing to check against.

Or do I?

Well, if the system date is correct, I really don't have to worry. In fact, I only need to be sure that the date isn't outrageously behind (if it's way ahead, that's the user's problem).

To prevent starting a program on a PC with a very stale date, I adopted the strategy of setting a base date and testing against that. I arbitrarily created the base date by using the date I finished the application, setting it up during the final make. *That* date was the base. So, I *can* test:

```
If ~CFG:FirstDate !no first date, must be first use
If Today() < Date(12,13,01) !date I started this article
```

If this condition is met, the date is hopelessly behind and I tell the user and terminate.

```
If ~CFG:FirstDate
  If Today() < Date(12,13,01)
    Message('Your system reports an invalid date/time.'  |
     'This could negatively impact your data.','Warning',|
      ICON:Hand)
    Return Level:Fatal
  End
```

```
End
```

If the date is current, the test will fail and I initialize the configuration variables:

```
If ~CFG:FirstDate
  If Today() < Date(12,13,01)
    Message(‘Your system reports an invalid date/time.’  |
     ‘This could negatively impact your data.’,’Warning’,|
      ICON:Hand)
    Return Level:Fatal
  Else     !later than base date
    CFG:FirstDate = Today()
    CFG:LastDate = Today()
    CFG:LastTime = Clock()
    Access:Config.Update
  End
End
```

If I have provided a copy of the `Config` file, with a blank record, the `Update` method is correct. Otherwise, `Access:Config.Insert` adds the new record.

If hard coding the base date into the program offends your sense of propriety (and it should), add another field to the configuration file, manually place a value in it and provide the file as part of your install. Then, the code above becomes:

```
If ~CFG:FirstDate
  If Today() < CFG:BaseDate
    Message(‘Your system reports an invalid date/time.’  |
     ‘This could negatively impact your data.’,’Warning’, |
      ICON:Hand)
    Access:Config.Close
    Return Level:Fatal
  Else
    CFG:FirstDate = Today()
    CFG:LastDate = Today()
    CFG:LastTime = Clock()
    Access:Config.Update
  End
End
```

This has the added advantage that, if someone should crack the encryption, unless that person is familiar with Clarion dates, the data will be less than useful (in fact, you should consider using misleading field labels to add to the confusion). Further, putting the base date into the configuration file allows you to periodically update the install without forcing you to remake the application.

Short take: this strategy ensures that the system date is not outrageously stale, that it is, at worst, relatively recent.

There are two further checks needed: first program start today and restarting a program.

Normal program start, morning

Assuming that the program has been started before (i.e., there is a `FirstDate`), a very similar strategy handles the first startup of the program on any given day:

```
If Today() < CFG:LastDate    !date rolled back?
  Message(‘Your system reports an invalid date/time.’  |
    ‘This could negatively impact your data.’,’Warning’, |
     ICON:Hand)
  Return Level:Fatal
Else        !update date and time of last use
  CFG:LastDate = Today()
  CFG:LastTime = Clock()
  Access:Config.Update
End
```

Not rocket science, is it?

Restarting a program

If a program has never been started before

```
If ~CFG:FirstDate
```

must be true and

```
Today() < CFG:BaseDate
```

shouldn’t be. And these two conditions tell you that you have a valid first start.

The first time the app is run on any given day,

```
Today() > CFG:LastDate
```

will be true and you know to update the last date of use. But, if an app is re-started later in the same day, none of the above conditions will be met. Instead,

```
Today() = CFG:LastDate
```

is true.

This is the most difficult case and it is the most important. This is the important case because time stamps and audit trails are unusable if this case cannot be resolved. In the case of timing an app out after a certain number of days, this is where you can trap a user trying to set the system date back.

Perhaps the user uses a single date over and over again. Fine. But what are the odds on their getting the time reset every time also?

At each date check, you read and store the time. If a date check failes, you don’t do anything further with the time. The same is true if the date is greater than the last use date. But, when `Today() = CFG:LastDate`, it follows that

```
Clock() < CFG:LastTime
```

cannot ever evaluate as true. If it does, either the system’s battery is failing, the system clock is seriously losing time or the user is changing the settings. There are no other explanations.

So, if `Clock() < CFG:LastTime`, the user gets the `Message()` and the program is terminated.

The final code to ensure, if not accuracy, reliability of the system date and time is:

```
Access:Config.Open
Set(Config)
Next(Config)
If ~CFG:FirstDate        !first use
  If Today() < CFG:BaseDate
    Message(‘Your system reports an invalid date/time.’,  |
     ‘This could negatively impact your data.’,’Warning’, |
      ICON:Hand)
    Access:Config.Close
    Return Level:Fatal
  Else
    CFG:FirstDate = Today()
    CFG:LastDate = Today()
    CFG:LastTime = Clock()
    Access:Config.Update
  End
Else                    !not first use
  If CFG:DaysAllowed    !if not 0
    If Today() > CFG:FirstDate + CFG:DaysAllowed
      !date-triggered action here
    End
  End
  If Today() < CFG:LastDate    !bad date check
    Message(‘Your system reports an invalid date/time.’,  |
     ‘This could negatively impact your data.’,’Warning’, |
      ICON:Hand)
    Access:Config.Close
    Return Level:Fatal
  Elsif Today() > CFG:LastDate  !first time today
    !fiscal year rollover check here
    CFG:LastDate = Today()
    CFG:LastTime = Clock()
    Access:Config.Update
  Elsif Today() = CFG:LastDate    !restart
    If Clock() < CFG:LastTime
      Message('Your system reports an invalid date/time.',|
        'This could negatively impact your ' &|
        'data.','Warning',ICON:Hand)
      Access:Config.Close
      Return Level:Fatal
    Else
      CFG:LastTime = Clock()
      Access:Config.Update
    End
  End
End
```

For applications that run 24 x 7, adapt and place in `Event:Timer`. And, for a little more protection, you could update `CFG:LastTime` when the program shuts down.

For the paranoid among us

Some time ago I wrote an app for an insurance company. I named the configuration file "Balances." They must have thought this was just one of the data files and they never touched it. However, the fact is that most users aren't going to try to crack the encryption of the configuration, or any other, file. But, if one tries there are two things that must be cracked.

To crack the file, first, the user must crack the encryption. While there are number of Clarion developers who can do this, it is not totally simple. Second, they have to crack the Clarion date and time algorithms. In other words, the user has to be ... motivated.

Some years ago, there was a technique described in the Clarion Tech Journal (I don't recall the author but the technique is quite devious) that may increase your comfort level. Create a second copy of the configuration file and place it in the root directory (or some place the user is not likely to be looking). Set the file's "hidden" attribute. Open both files and compare the values in each field. If they do not match (or, if you transform the values in one, do not match expectations), one of the files have been hacked. Terminate the user … I mean the application.

When updating values, of course, write to both files.

Summary

Having control over the system date and time is often necessary to provide features like audit trails, time limited usage or other date sensitive functionality. Enforcing clock accuracy just is not possible. But, while you have no protection against a clock that is perpetually off or users who re-set clocks, that does not mean that you cannot ensure that date and time increment as expected – even if it takes manual intervention by the user.

This is just another one of those cases where defining what you really need makes the code easy. The code may not be rocket science, but the planning is.

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitors' right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Reader Comments

Add a comment

Hi Steve, Good article, maybe instead of seperate date...
Greg, EXCELLENT idea. This would also simplify the...

Clarion Magazine

For marketing your Applications
and Developer Accessories or to
purchase other 3rd Party Tools . . .
Developer PLUS tm

A Template For Copying Fields Between Files

by Andrew Guidroz II

Published 2002-01-10

Assignment statements are what programming is all about. Programs move data from one field to another constantly. If word processing and number processing (spreadsheets) are what drove the PC revolution in everyone's small office, file processing (moving data among various files) is what drove most of us to using Clarion.

One of the biggest complaints I hear when folks move from DOS Clarion to Windows Clarion is that you cannot do columnar block copies in the editor. Why is this so important? Folks use it in order to write blocks of assignment statements. But don't you wish the assignment statement was smarter?

In fact, there is a very impressive feature within Clarion known as the deep assignment. From the online help:

> The :=: sign executes a deep assignment statement which performs multiple individual component variable assignments from one data structure to another. The assignments are only performed between the variables within each structure that have exactly matching labels, ignoring all prefixes. The compiler looks within nested GROUP structures to find matching labels. Any variable in the destination which does not have a label exactly matching a variable in the source, is not changed.

So you can do the now famous record copy this way:

```
NewPrefix:Record :=: OldPrefix:Record
```

Just think of all the typing that saved! And think of the time it saved! And I hate it.

Here are a pair of group structures for you:

```
NewGroup GROUP
Field1
Field2
Feild3
         END
```

```
OldGroup GROUP
Field1
Field2
Field3
         END
```

Say I write the following:

```
NewGroup :=: OldGroup
```

There is a problem. The first two field assignments work just fine:

```
NewGroup.Field1 = NewGroup.Field1
NewGroup.Field2 = NewGroup.Field2
```

However, the third assignment fails, because I misspelled the word FIELD in NewGroup:

```
NewGroup.Feild3 = ''
```

Of course, I probably won't realize this has happened because the deep assignment operator is a black box.

Then there is the problem of adding a field to the old group and forgetting to add the field in the new group. I hate it when that happens. And what if my assignment needs something just a bit different? What if what I want to do is make the value of each numeric field in the new record one more than the value in the old record? I'm back to doing all that typing again.

As well, Clarion's large, complex IDE, as good as it is, is not without its own bugs. In-place conversion of a data file has been known to fail for various reasons throughout the years. The most recent from the 5.5.07 release candidate:

> *6. Browser*
>
> FIX: If file is declared in the dictionary with a variable password, in-place conversion can fail.

Now, who knows if that means the entire process will fail or just some of the assignment statements? There must be a better way.

I finally wrote my own template for assigning data from one file to another. It requires a file to copy from and a file to copy to. This template is a code template that can be placed in any embed point, such as procedure routines, within a method, etc. It generates matching assignments for each field that has the same name. It also generates a comment line for every field that does not have a match so you can look at the generated code and decide for yourself what you want to do with the "extra" fields. There is also an option to generate all code as comments so you can use just those lines you want by copy and pasting them. Here's the code that creates an assignment for matching fields:

```
#FOR(%NewFileFields)
    #FOR(%OldFileFields)
        #IF(UPPER(CLIP(%NewFileFields)) ↵
           = UPPER(CLIP(%OldFileFields)))
 %AllCommentsString %NewFilePrefix:%NewFileFields ↵
    = %OldFilePrefix:%OldFileFields
            #BREAK
        #ENDIF
    #ENDFOR
#ENDFOR
#!
```

Note the use of the %AllCommentsString variable. The template has an option to generate all code as commented so you can review the code without inadvertently running it. If you want the code commented, %AllCommentsString is set to ! instead of an empty string.

The following code checks for missing fields in the old file (there is a similar block for finding missing fields in the new file. In each case the template generates a comment informing you of the missing field.

```
#FOR(%NewFileFields)
    #SET(%GotOne,'False')
    #FOR(%OldFileFields)
        #IF(UPPER(CLIP(%NewFileFields)) ↵
          = UPPER(CLIP(%OldFileFields)))
            #SET(%GotOne,'True')
            #BREAK
        #ENDIF
    #ENDFOR
    #IF(CLIP(%GotOne)='False')
 ! There is no old field for this new field.
 ! %NewFilePrefix:%NewFileFields = ! ????????
    #END
```

```
#ENDFOR
```

I typically use this template within the TakeRecord method of the ProcessManager of a Process Window in order to handle file conversions. I place it in an embed in the Procedure Routines sections of a procedure with an embed just prior to it declaring it as a routine so I can use it thoughout my procedure. I also have it wrapped up within an extension template that does more complex file conversions (prompts for variable file names, etc). I find it very useful and I hope you do too.

Download the source

Andrew Guidroz II, when he isn't traveling around the countryside watching his 2001 SEC Champion LSU Fighting Tigers, writes software for all facets of the insurance industry. His famous Cajun cookouts have become a central feature of Clarion conferences throughout the U.S. Andrew's Cajun website is www.coonass.com.

Reader Comments

Add a comment

Clarion Magazine

Handing COM Events - Part 1

by Jim Kane

Published 2002-01-11

The one area of COM that the Clarion OLE control is particularly weak in is receiving events. When the OLE control works, it works very well and is very simple to use. However, more often than not, it does not work. One such case is with Microsoft Office products. If you want to get an event notification from Outlook when mail arrives using the Clarion OLE control, you're out of luck.

Some time ago I wrote an order processing system that relied on sales people emailing orders. The orders were received and processed manually. Then the company wanted to process the emails automatically. This meant detecting when new mail arrived. Outlook was simply going to have to tell me when new mail arrived, as it seemed horribly inefficient to constantly enumerate the inbox. Clearly it was time to see how COM events work and teach Clarion some new tricks. In this article I'll build on information I've presented in other articles, here in Clarion Magazine.

To determine if a COM object can generate events, open the object or its type library in OLEVIEW. For Outlook, navigate to your Microsoft office/Office folder and locate a file called MSOUTL9.OLB. The number may vary depending on the version of office. I currently have version 9 of Outlook 2000. Once the type library is displayed in OLEVIEW search the left hand pane for `CoClass` entries in the tree. Find the application `CoClass` and open it. On the right hand pane you'll find the following:

```
[
  uuid(0006F03A-0000-0000-C000-000000000046),
  helpcontext(0x004de932),
  appobject
]
coclass Application {
    [default] interface _Application;
    [default, source] dispinterface ApplicationEvents;
};
```

Notice the `[default,source]` just before `ApplicationEvents`. Interfaces with `,Source` indicate they are 'source' or generate events. By examining the `ApplicationEvents` dispatch interface (`dispinterface`) you can determine what events the Outlook application object generates. Some objects will have many source interfaces and can generate many different events.

While not directly needed for or related to events, the `uuid` above is also called a `CLSID` and identifies the Outlook Application Object. This `CLSID` can be used to create an instance of Outlook, or attach to a running instance of Outlook.

Now use the tree in OLEView and drill down to the `ApplicationEvents` interface. This is what appears on the right side:

```
[
  uuid(0006304E-0000-0000-C000-000000000046),
  helpcontext(0x0053ec60)
]
dispinterface ApplicationEvents {
    properties:
    methods:
        [id(0x0000f002), helpcontext(0x0050df84)]
        void ItemSend(
                        [in] IDispatch* Item,
                        [in] VARIANT_BOOL* Cancel);
        [id(0x0000f003), helpcontext(0x0050df85)]
        void NewMail();
        [id(0x0000f004), helpcontext(0x0050df86)]
        void Reminder([in] IDispatch* Item);
        [id(0x0000f005), helpcontext(0x0050df87)]
        void OptionsPagesAdd([in] PropertyPages* Pages);
        [id(0x0000f006), helpcontext(0x0050df88)]
        void Startup();
        [id(0x0000f007), helpcontext(0x0050df89)]
        void Quit();
};
```

There are two important numbers to get from this screen. First, COM identifies interfaces by universally unique identifiers, here shown as a `uuid`. Commonly a `uuid` that identifies an interface is called an `IID` or interface id. In any case, copy it to an equivalent Clarion group like this:

```
!0006304E-0000-0000-C000-000000000046
syncIID        group
```

```
data1              ulong(06304EH)
data2              ushort(0)
data3              ushort(0)
data4              string('<0C0H><0><0><0><0><0><0><46H>')
   end
```

The particular event of interest is `NewMail`. Again rather than referring to an event by its name, COM refers to it by its `DISPID` or `ID`. The `DispID` for `NewMail`, in Clarion format, is `0F003H`.

That is all the information (`IID` and `DispID`) needed to connect to Outlook and have it send events to a Clarion program.

The steps to connect a Clarion program to a COM object, in this case Outlook, are:

1. Initialize COM – `CoIntialize()` API
2. Create an instance of Outlook, or attach to a running instance.
3. Ask Outlook if it supports events, and in particular the event interface called `syncIID`. Unless it's an older version of Outlook, it will.
4. Tell Outlook what Interface and procedure in your program to call when it has an event. In COM terminology this is 'Advise' or 'Connect'.
5. Store any information (parameters) that came with the event in global memory
6. Send a message to a Clarion window that an event was received and the information is available in global memory.
7. Process the event reading the global memory
8. When done, disconnect, or in COM terminology 'Unadvise'
9. Uninitialize COM

All the steps above are shown in the sample code that comes with this article. That code attaches to a running instance of Outlook and monitors it for mail. To see it in action, start Outlook, then compile and run the sample project `outl.exe`. As soon as Outlook receives mail, you'll see the message box pop up. In real life you could then retrieve the email message or take any other appropriate action.

In a previous series of articles I introduced a class called `StdComCl`. This class can do many of the things needed for this project, including step 1 and 9 plus a lot of error handling. All that `StdComCl` is lacking is the event-specific code. To make maximum use of `StdComCl` I decided to derive a new class from it called `ConPtCl`, which is short for "connection point class." The downloadable code that accompanies this article includes both StdComCl and the derived `ConPtCl`.

Step 1 using the `StdComCl` is very simple and is covered in more detail in earlier articles. All that is needed is a call to `ConPtCl.InitCOM()`. This method calls `CoInitialize()`.

Step 2 involves calling the API `GetActiveObject()` like this:

```
getactiveobject(address(clsid:outl),0,lpobj)
```

The `clsid` is another COM identifier in this case for Outlook. You can get it out of the Outlook type library using OleView. The output is `lpObj`, a long pointer to the address of the running Outlook instance, or more specifically the address of its `Iunknown` interface. The raw address is converted to an interface with the following line of code (see the previously mentioned series of articles for details).

```
IUnk&=(lpObj) !create the iunknown interface
```

If Outlook isn't running, the call to `getActiveObject()` will fail, `lpObj` will be zero, and the code just closes down gracefully.

Step 3 is handled by the `Connect` method of `ConPtCl`. It calls the query interface method of the `Iunknown` interface obtained in step 2 and asks if the object supports the `IconnectionPointContainer` interface. If so, `Connect` asks the `IconnectionPointContainer.FindConnectionPoint()` method if it supports the interface (`syncIID` above) that is required for this code. If any of this fails, the `Connect` method returns `level:fatal` and the program closes gracefully.

Step 4 is the most important. The code now tells Outlook where to call when it has an event. The address to call is passed to the `advise()` method of the `IconnectionPoint` interface obtained in step 3. The problem is Outlook expects to call a COM `Idispatch` interface. How do you make one of those? Fortunately it's pretty simple.

All you need to do is create a class that implements an `Idispatch` interface. The `CWDSyncCl` class in the accompanying code does just that. Most of the methods simply return `0` or `S_OK`. The only method of interest is `Invoke`, which receives the messages sent from Outlook. When `Invoke` is called, this method receives the `dispid` (`0F003H` is the one of interest in this project) and an array of parameters. I covered the details of how `Idispatch.Invoke` handles parameters in a previous article. The code here takes the `dispid` of the event received and its parameters and passes it to `CWDSyncCl.TakeEvent()`. In many cases you'll want to derive `CwDsyncCl` and add some code to the `TakeEvent()` method. In the sample code, the only event of interest has a `dispid` of `0F003H` so to filter out all other events you do this:

```
cwdsynccl.takeevent procedure(long dispID, |
    long Paramcount, long lpParam, *long lpResult)
  code
  if dispid<>0F003H then return.
  Parent.TakeEvent(dispid,paramcount,lpparam,lpresult)
  Return
```

The `parent.TakeEvent()` method has two important jobs. First it has to put any parameters for the event pointed to by `lpParam` into global memory (memory allocated by the win32 API `GlobalAlloc()` call). `TakeEvent` then sends a message to a Clarion window to notify the window that an event was received. One thing to keep in mind is when `CWDsynccl` is called it is being called by Outlook and *not* by your Clarion program. The thread that accesses it belongs to COM and/or Outlook. Both the calling COM object and the Clarion Window access the global memory. If both were to access it at once, errors would be sure to occur. To avoid that, a critical section (which I've written about in another ClarionMag article series) is used to allow only one or the other to access the global memory at one time. All the member variables are private and only accessed when the critical section allows it. The code that does this is in blockCl.inc/blockCl.clw and is quite simple. If you have not use critical sections much you may want to take a quick look to convince yourself there isn't much to them. In order to keep things simple, I let `ConPtCl` set up `blockcl` and choose various block sizes and global memory size. The default settings allow for up to 15 parameters to accompany any event. While I doubt that will ever be a problematic limit, it is something to keep in mind. Conversely if you are willing to accept a lower limit for the number of parameters, you could reduce memory requirements drastically.

The parameters passed with events are passed as variants. I covered variants and the details of how a dispatch interface passes parameters in a previous article. One complexity in copying variant parameters to the global memory is some variant types like BStrings contain an address rather than the data itself (data passed by address rather than by value). This complicates matters because the data pointed to by the address in the variant may be destroyed before you process the event. To avoid that problem you can use a special API call, `VariantCopyInd()`, which copies not only the variant structure but also what the variant points to. Later you will call a class method called `variantFree` to free this memory allocated by copying.

That's all for this week – next week I'll explain how to notify the Clarion program that an Outlook mail event has happened.

Download the source

Jim Kane was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and after graduating served in the US Air Force. He is now retired from the Air Force and writing software for ProDoc Inc., developer of legal document automation systems. In his spare time, he runs a computer consulting service, Productive Software Solutions. He is married to the former Jane Callahan of Cando, North Dakota. Jim and Jane have two children, Thomas and Amy.

Reader Comments

Add a comment

Hi Jim, Just to let you know, the Code is not thread...
Hi Jim, What version of OleView do you use. Mine won't...
Rick - I think that's the current version (see...
Thanks, Dave. You're right 2.1.057 is the latest at the...

The ClarionMag Sweepstakes January Draw Is Coming Soon!

Published 2002-01-14

Enter the Clarion Magazine Sweepstakes and **you could win** one of the following:

First Prize (winner to choose one)

- a **Compaq iPAQ** Pocket PC, **or**
- a **registration** to the **ETC** Clarion Conference in Gatlinburg in May, 2002

Additional Prizes

- a **two year** Clarion Magazine subscription or renewal, value $150
- a **one year** Clarion Magazine subscription or renewal, value $80
- five **six month** Clarion Magazine subscriptions or renewals, value $40 each (already awarded)
- a copy of **CW Assistant**, valued at $99 (already awarded)
- two copies of the **Clarion Source Search** utility, valued at $45 each. (already awarded)
- a copy of G-Cal, valued at $99 (already awarded)
- a copy of G-Calc, valued at $69 (already awarded)
- a copy of G-Buddy, valued at $99 (already awarded)

Note: all of the above subscriptions/renewals can also be taken as back issues, in whole or in part

Link to this Sweepstakes Page!

Add a link to this page on your web site, and email dharms@clarionmag.com - we'll add an exchange link here.

Sweeps Sponsors

- Carl Barnes Computer Consulting
- Gitano Software

Sweeps Partners

- Encourager Software
- Brady & Associates

How To Enter

There are four ways you can enter the Clarion Magazine Sweepstakes:

- Refer a friend to Clarion Magazine (you supply your friend's name and email address, and we send your friend a one-time only email); or
- Take out a new subscription to Clarion Magazine (for a free sampler click here); or
- Renew your existing Clarion Magazine subscription; or
- Mail a handwritten postcard (see the rules for details).

That's all there is to it! The sweepstakes ends February 28, 2002, so get your entry (or entries) in now! And start canvassing your friends and co-workers - remember, for every person you refer to Clarion Magazine, you get an entry in the sweepstakes. The more entries you have, the better your chances!

Rules

The official rules are available here.

Reader Comments

Add a comment

Is it true this contest ended last January 15, 2001? Sweepstakes ends January 15, 2002! So much for my...

Linder SetupBuilder 4.0 LINDERSOFT
Everything Else is Just Vaporware

Eleven Winners In ClarionMag Sweeps First Draw

Published 2002-02-01

The following people are winners in the Clarion Magazine Sweepstakes interim draw:

- Six month Clarion Magazine subscription/renewal: **Eric Griset**
- Six month Clarion Magazine subscription/renewal: **David LeYanna**
- Six month Clarion Magazine subscription/renewal: **Gary Stanley**
- Six month Clarion Magazine subscription/renewal: **Uro Mencinger**
- Six month Clarion Magazine subscription/renewal: **Nick Tsigouro**
- CW Assistant utility, valued at $99: **Ramon Reed**
- Clarion Source Search utility, valued at $45: **S. Hills**
- Clarion Source Search utility, valued at $45: **Janice Cournoyer**
- G-Cal, valued at $99: **Chantal St. Jean**
- G-Calc, valued at $69: **Sherae Gronbach**
- G-Buddy, valued at $99: **Antonio Oliveira**

The above-named have been notified by email. All winners (and all others who have entered the Sweepstakes) are eligible for the final draw at the end of February. The grand prize is the winner's choice of a Compaq iPAQ or an ETC-III conference registration. See the sweeps page for details.

Reader Comments

Add a comment

Clarion News

Published 2001-11-21

INN Bio: Mihai Palade

This week's INN bio subject is Mihai Palade, a Clarion programmer from Romania. A recent software engineering graduate, he's now studying for his Master's in Measurement Systems at the "Politehnica" University of Bucharest. But at his day job, he works with Clarion. Read about the surprising (maybe surprising to some) way he was introduced to Clarion, and see a little of his corner of the world.
Posted Thursday, January 31, 2002

Advertise On The Clarion Connection

The Clarion Connection now accepts paid advertising for products, services and events related to Clarion. The right frame shows four ads, chosen randomly each time the page is seen. The rate is 0.03 (USD) per view (1000 views minimum charge). Your advertisement can be managed online using a secure server to change the graphic, link, alt-text or to make payments using MasterCard or Visa. Other payment methods can be arranged.
Posted Wednesday, January 30, 2002

Digital Alarm Clock Demo App

Ville Vahtera has uploaded a new "digital" clock demo to Steve Parker's download center. This app is freeware.
Posted Wednesday, January 30, 2002

New Icetips Previewer Demo

A new Icetips Previewer demo is now available. The Standard Demo reports demonstrate "smooth" scrolling through pages using the PageUp and PageDown keys. They also show printing the current page without leaving the Previewer, which can be very handy. The Page of pages report demonstrates a way to support this feature in the Icetips Previewer. You can now get the Icetips Previewer and the

Icetips Reporter in a bundle for 199. The Icetips Wizards + Icetips Previewer bundle is also $199. After January 31st the price for both bundles goes to $249.
Posted Wednesday, January 30, 2002

New Home For Schoeffler & Rau Datensysteme

The Schoeffler & Rau Datensysteme web site now has a new home at www.schoeffler.biz/cw.html. Please update your links.
Posted Wednesday, January 30, 2002

SAPDB Available For Download

Kelvin Chua reports that version 7.3.00.20 of SAP DB Software is available now for downloading.
Posted Wednesday, January 30, 2002

Icetips Previewer Released

Icetips Software has released a fully-customizable report previewer. The Previewer Wizard creates a previewer procedure in your application. This is just a normal window procedure, with all the embedded code available for modification. Since the previewer is an app procedure, you can have multiple previewers in your application. Features include: extension template to apply previewer to reports; page list; fast search; easy viewing, deleting, and printing of pages. The Icetips Previewer is available through January 31, 2002 for US$99.00. As of February 1, the price goes up to $149.00. Compatible with Clarion 4, Clarion 5, Clarion 5.5, ABC and Legacy as well as the CPCS reporting templates in both Legacy and ABC. The Previewer demo includes four different previewers. Until February 1 you get the Icetips Checkbox fixer thrown in as a bonus with any Icetips product or combination you buy.
Posted Monday, January 28, 2002

SealSoft xXXL Pack

SealSoft has a 15-product bundle available for $740.
Posted Monday, January 28, 2002

DateTime Clock Special Pricing Ends Thursday

Special pricing (33% discount) for the Sylvan Computing clock/calendar classes ends on January 31, 2002. These classes include: Date keys - use quicken style keys to enter modify the dates. (-,+ Move date by one, CTRLT = Today() etc.); Calendar dropdown - a 100% Clarion control that mimics the mscal.ocx; Clock dropdown - also a non-modal drop down window. Demo available. Binary version is now US$20.

Posted Monday, January 28, 2002

CPCS For C55 Now supports Internet Connect Under ABC

CPCS v5.50d has now been modified to support Internet Connect under ABC-based applications. This is in addition to Web Builder, which was already supported under ABC. Only Internet Connect is supported under Legacy applications. Support for WB or IC is activated automatically by adding the appropriate SV web template.

Posted Monday, January 28, 2002

CPCS V5.50d Previewer Supports Prompt, Tip, And Icon Customization

The CPCS Help and PDF files for v5.50d have been modified to better document the ability to customize icons in the Previewer. This capability have been in CPCS for quite some time, but the help and PDF file text did not specifically mention that icons could be customized (although there were examples of doing so in the help/PDF files).

Posted Monday, January 28, 2002

New Icetips Previewer Demo

Arnor Badvinsson has uploaded a new (900k) demo of the Icetips Previewer. This demo uses two previewers, but instead of previewing a report, it is an imageviewer. It has one basically standard Icetips Previewer, the other one modified a bit to preview a queue created with the Directory() function. You can view bmp, gif, jpg, jpeg and pcx files there, and print them. Of course when you print from the image viewer, it calls the other Icetips Previewer first.

Posted Monday, January 28, 2002

New Wizard Template And Classes Released

Sylvan Computing has released scWizard, a tool that helps create a Wizard form. This extension is applied to a window that has a SHEET, Next Button, Back Button, And a Finish Button. Once populated there is no code you need to write to provide navigation through the wizard dialog. There are virtual methods provided so the programmer can capture the event of changing wizard pages and for validation. scWizard is US$30.

Posted Monday, January 28, 2002

DOS Printer v5 Released

Dave Beggs has just released DOS Printer version 5. This utility allows you to print from DOS to any Windows printer. New in this release: you can specify wildcards in the filenames - i.e. set it to print c:\temp\*.txt; you can specify the left margin (for those laser printers that can't print close to the paper edge). DOS Printer is

US$19.99.
Posted Monday, January 28, 2002

AnalyZe.IT
AnalyZe.IT lets you make a Management Information System from your application data, creating 2/3D graphics or tabular reports. You can distribute reports with Preview.IT. Features include: RunTime Variables; various formats including PDF, XML,HTML,RTF,LL etc.; report scheduling; report emailing; report FTP.
Posted Monday, January 28, 2002

Dennis Evans' Web Site
Dennis Evans has posted some coding notes on the web. Currently these include a brief discussion of using a derived WindowManger and the register function to enable/disable menu options. Future updates will depend completely on the usage.
Posted Monday, January 28, 2002

Clarion TWAIN Demo And Template
Ville Vahtera has posted a TWAIN demo using the freeware EzTwain library with Clarion. The included template currently only handles the library part, not the blobs. The app demonstrates using the library as well as how to place scanned images into blob file and restore them back. The cwtwain.exe file can be downloaded from Steve Parker's download center.
Posted Monday, January 28, 2002

SealSoft xWinSet v2.09 Released
The latest release of xWinSet (2.09) contains a fix to objects blinking on a window with a large number of controls. New demo and install available.
Posted Monday, January 28, 2002

Clarion Handy Tools Thin Client Data Demo
In October Clarion Handy Tools added Browser Data Server Technology. This is a set of templates and classes (part of the standard CHT toolkit) that help you build a Clarion application that can act as a data server to any HTML 4.0 capable browser. Data queries are entered from a "Query Page" and the result set is returned to you in the form of an HTML data table. Now the CHT includes a thin client application that can connect to this same "Browser Server" application and display data in a more traditional desk top application format. This browse is hard to tell from any of our browse demo applications; the difference is that the data base is remote and that data are served up as you do queries on the browse query control. The app also has a configurable data buffering capability which for a user defined period of

time (default 5 minutes) maintains a copy of all query result sets should the same query be repeated within the time limit. Demo available at http://www.cwhandy.com/pcdemos/hndhtgtdemo.exe (960KB).
Posted Monday, January 28, 2002

SysDTP Date And Time Picker

SysDTP is a Date and Time Picker from solid.software. This is another in the company's series of wrapper classes for common controls of the WIN32 API. SysDTP is the drop-down calendar control you'll know from applications like Outlook Express or even the Windows Explorer. This calendar control implementations gives your application the native Windows look and feel. Features include: Drop-down calendar for date entry; Up/down controls for time entry; Optional checkbox to allow empty dates; Fully customizable fonts and colors; Internationalization from control panel or user format strings. SysDTP supports both ABC and Legacy templates in applications compiling to 32bit only. LIB and DLL versions available for standalone and local runtime libraries. Demo available. SysDTP is US$39 from ClarionShop.
Posted Wednesday, January 23, 2002

xNotes v1.2 Released

This release of xNotes contains just one bug fix - reminder popups in a minimized program could result in the program hanging. New demo and install available.
Posted Wednesday, January 23, 2002

Dennis Evans Featured INN Bio

This week, the Icetips News Network profiles Dennis Evans, of one of San Antonio's newest residents.
Posted Tuesday, January 22, 2002

Gitano G Build Support

Gitano Software customers that are using the G build can now get new downloads for Gitano products. These are not public releases and will not be until SoftVelocity officially releases the G build. Registered user can obtain the link for each utility by accessing the Gitano support forum (you must be registered to access the developers forum), or email.
Posted Tuesday, January 22, 2002

Free Data Dictionary For A Software Project Management System

Greg Berthume has a free data dictionary that contains a number of project management files/tables already created and related. Just start building the app and add whatever third party add-ons you desire. Custom solutions also available.

See the downloads page.
Posted Tuesday, January 22, 2002

Shareware Version of EmailData Template ABC/Legacy Available

A shareware version of Vivid Help's EmailData Template for ABC/Legacy is now available. Please remember that shareware version and only the shareware version needs the EXE file in your working directory to work properly.
Posted Tuesday, January 22, 2002

CPCS Preview Now Handles Varying Sized Pages

CPCS v5.50d is now available. This new build has a modified CPCS Preview which accommodates previewing of different sized pages in the same report. This change is implemented to more fully support the Concatenate Reports template recently added to Feature Enhancement Set# 1. Current users of CPCS v5.50d can install this new build using their current authorization codes.
Posted Tuesday, January 22, 2002

Freeware Capitalize Template Updated

Sterling Software's freeware CapFlash 2 template now includes an ABC demo app. CapFlash is an extension template to be used on a Process - it will convert a file of all upper case to Proper Case ("capitalize") with the following options: You can enter into the template a list of words which are always lower case - such as del, la, de etc.; There is also a user-defined list of words which are always UPPER case - such ABC, USA, AFB etc.; Individual fields (such as State) can be excluded; Names beginning with Mc,Mac or O' are also provided for. CapFlash is compatible with CW2002 to C5.5, ABC and Legacy.
Posted Tuesday, January 22, 2002

RAS (Dial-Up) Library Freely Available

Ville Vahtera's free RAS library is now available from Steve Parker's Par2 web site. Install includes library, demo app, and template. The demo is made for Clarion 5PE and up, but you can use the template with Clarion 4 and later.
Posted Tuesday, January 22, 2002

xWord Library v1.6 Released

New in xWord Library 1.6: you can now set TemplateFile and DataFile before executing MergeDocuments; there are four new methods (MergeSetTemplate, MergeSetDataFile, MergeGetTemplate, MergeGetDataFile). New install, demo, and docs available.
Posted Wednesday, January 16, 2002

Handy Tools Example Apps Revised

The Clarion Handy Tools currently ship with 34 example applications. These are being checked and revised for 2002 to reflect the latest capabilities of the toolkit. Applications include: FTP client; programmable FTP engine (push files to or pull files from a remote FTP server); email inbox reader using Outlook or Outlook Express; email client; email address extractor and list manager; bulk mail sender; internet/intranet data server; extensions to the ABC browse including header-click browse column sorting, built-in locators, greenbar support, etc.; record marking; various window capability extensions; and more.

Posted Wednesday, January 16, 2002

EmailReport ABC Version 1.1

A new version of VividHelp's EmailReport template is now available for download. New features include: EmailTo now can be populated from a variable, so you can send reports to as many addressees as needed; report pages can be resized in the email message body, making it possible to fit big reports into the email.

Posted Wednesday, January 16, 2002

Free EFT/Credit Card Template Beta

Andy Stapleton is looking for beta testers for a new template/DLL for handling EFT (ACH) and credit card transactions via www.paywire.com. PayWire is a billing service company handles recurring business transactions and also provides secure FTP upload and collections processing.

Posted Wednesday, January 16, 2002

New Clarion Site

Kelvin Chua has moved the contents of www.accpro.com.sg/clarion to www.clarionpost.com. The new site includes user-updateable Clarion links.

Posted Monday, January 14, 2002

EnhancedScrollClass C3PA Compliant And C5.507 Compatible

ThinkData has retooled the installation of the EScroll to be C3PA compliant with respect to the installation directories. It is also fully compatible with the latest Clarion 5.507 release. An updated demo is available.

Posted Monday, January 14, 2002

CPCS Adds Report Concatenation Feature

CPCS has added a new Report Concatenation feature to its Creative Reporting Solutions product. This new feature allows you to concatenate (append) multiple

CPCS reports to create a single "final" report which can then be previewed, printed, and/or (via other CPCS addons) faxed, converted to PDF, or emailed. This new feature has been added to the CPCS Feature Enhancement Set #1 addon which can be purchased and downloaded directly from the CPCS website. Current owners of this addon can download the updated install file and use their current authorization codes to install this new build free of charge.
Posted Monday, January 14, 2002

New TeamIDD Pricing Structure

TeamIDD, a distributed source control application, now has a new pricing structure. To register a project, regardless of the number of developers or files involved is 25.00 USD, free until Jan 18. To check in a file is 0.25 USD, except for the first 5 days when there is no charge. To add a developer to a project is free. To download the client program to your computer(s) is free. TeamIDD communicates with its host computer using the Secure Socket Layer where all communication is encrypted; the TeamIDD server is hosted at a data center owned by Xodiax, one of only five IBM certified data centers in the USA. Note: If you downloaded the client program, please download the new, secure client. The old client will be phased out.
Posted Monday, January 14, 2002

Gitano Software Support Forum

A new support forum is now available for all Gitano Software customers. There are two boards, 'General', and 'Developers'. You can log in anonymously to the 'General' board, but access to the 'Developers' board requires registration.
Posted Monday, January 14, 2002

Upgrade To A Gitano Bundle Promo

Upgrade your Gitano Software utilities to a bundle and Gitano will deduct 100% of your original price from the bundle price. If you do not have current versions of the utilities you must first upgrade to them. To purchase, simply order the bundle at the regular price and Gitano will deduct the difference before charging your card. This offer is good until Feb 15th 2002.
Posted Monday, January 14, 2002

DOS Printer version 4 Released

DOS Printer is a C5b ABC program which prolongs the life of CPD 2.1 programs. It sits in the system tray and monitors for the presence of a specified text file. When it finds it, it converts the ANSI line type characters to their nearest courier equivalent and prints it to a pre-set windows printer. If you have DOS clients who have trouble

printing to USB printers, or non-DOS compatible printers, or who would like to print to fax or PDF type drivers then check it out.
Posted Monday, January 14, 2002

xQuickFilter v2.09 Released
New in xQuickFilter 2.09: a small bug fix regarding entries containing single quotation marks; option to assign a hot key which will assign a hot field to another variable. New demo and install available.
Posted Thursday, January 10, 2002

Sterling Data New Year Bundle Discounts End Jan 10
Sterling Data is offering New Year bundle discounts of up to 40% on most of its products. This offer ends January 10, 2002.
Posted Thursday, January 10, 2002

Juan Domingo Herrera Featured In INN Bio
The subject of this week's Icetips News Network Bio is Juan Domingo Herrera of Buenos Aires, Argentina.
Posted Thursday, January 10, 2002

Compile Manager 2 Available
A new release of Gordon Smith's Compile Manager 2 (release 5.005) is now available. This is a 32 bit version.
Posted Monday, January 07, 2002

Translator Plus Legacy Template Released
ProDomus has released Translator Plus Legacy Templates for C5 and C55. These apply Translator Plus Class Libraries to Legacy applications. The package includes modified standard templates that make it possible to translate the many otherwise inaccessible strings generated by the legacy template chain. It also includes a redirection file to use these templates without the need to modify the standard installed files. The release includes a new class that can be used to pop up a user selection window. While there are many ways to initialize the selection of a language files and locale for picture translation, this provides a very simple-to-implement starting point. Users of Translator Plus may upgrade to this version at no charge. Users of PD Translator or CW Intl may purchase the Translator Plus editions at a 50% discount.
Posted Monday, January 07, 2002

TeamIDD Distributed Source Control

Industry Data Design has released its TeamIDD multi-developer distributed source control package. TeamIDD works over the internet, and handles all files, not just .APP and .DCT files. To register a project, regardless of the number of developers or files involved is 25.00 USD, or no charge until Jan 18. To check in a file is 1.00 USD, except for the first five days when there is no charge. There is no charge to add a developer to a project, or to download the client program to your computer(s). When you check in a file, the client program transmits it to the IDD server, where it is stored in a database in IDD's constantly staffed, redundant and fullly backed up data center. Your teammates see the new version is available within 60 seconds (if they happen to be in). In a single operation, TeamIDD will retrieve all the files that are out of date on your teammate's computers.
Posted Monday, January 07, 2002

Shareware EmailReport ABC
A shareware (try before you buy) version of EmailReport ABC template is now available from Vivid Help.
Posted Monday, January 07, 2002

RInstall Update
RInstall has been updated to version 1.b, and is available for download using a new password (if you are a user and have not received an email regarding the update, email bdl@riebens.co.za). The update adds two new embed points and also fixes a problem with the seed key for the serial number and unlock code generator.
Posted Monday, January 07, 2002

Fomin Report Builder Public Support Forum
There is now a public support forum for Fomin Report Builder. This enhanced support resource is designed to replace the old technical support practice of emaling the author directly. This is a public support forum; anybody may ask question and anybody may answer using web interface. Only "help", "how to", "is it possible" or similar questions regarding the product are allowed. A FAQ section is also available.
Posted Friday, January 04, 2002

PSI's TimeTrak 1.1 Released
PSI's TimeTrak 1.1 has been released. This version cleans up some minor bugs and adds some additional reporting capability. Plus it includes the ability to export billable time. Export has a sister feature, import. With the two it is easy gather timesheets from subcontractors and consolidate the information into one billing report to the client.
Posted Friday, January 04, 2002

INN Bio For Jan 3, 2002

In our first biography of the new year, the Icetips News Network is pleased to present an interview with another well-known Clarion developer. He has been a Clarionite for quite awhile, was once a third-party developer, and is still active on the technical newsgroups. Find out what he likes best about being a consultant and see some great photos of northern California and beautiful examples of his handiwork.

Posted Friday, January 04, 2002

New Scripting Language For Clarion And Business Users

In January QD Software will releasing a beta version of a scripting language written in Clarion. While this tool has been developed for business users it has also some options that will be of use to Clarion Developers. This scripting tool includes support for: data dictionaries; scripting code generation; application frames - MDI, toolboxs, dockable toolbars; database services (RI and field validation, group breaks SQL support); run time window creation; scripting methods and subroutines; program control statements; field object creation; browses; EIP. Browse and list box control are separated for future development; additional features are planned.

Posted Friday, January 04, 2002

ForeHelp Closes Doors

After seven years of serving the Help authoring community with ForeHelp products, ForeFront is closing its doors. ComponentOne plans to integrate some key ForeHelp technology into an upcoming release of Doc to Help. ForeFront will ship product through Thursday, January 10 at a special 60% discount. Payment must be made at time of purchase with a credit card. Effective immediately, there is no technical support for ForeHelp products.

Posted Wednesday, January 02, 2002

Reader Comments

Add a comment

Clarion Magazine

Handing COM Events - Part 2

by Jim Kane

Published 2002-01-16

Last week I began explaining how to get a Clarion program to receive events from an OLE control. When the Clarion OLE control works, it works very well and is very simple to use. However, more often than not, it does not work. I had a particular need to notify a Clarion program when Outlook received an email. Building on work I've previously discussed in Clarion Magazine, I took the following steps:

1. Initialize COM – `CoIntialize()` API
2. Create an instance of Outlook, or attach to a running instance.
3. Ask Outlook if it supports events, and in particular the event interface called `syncIID`. Unless it's an older version of Outlook, it will.
4. Tell Outlook what Interface and procedure in your program to call when it has an event. In COM terminology this is 'Advise' or 'Connect'.
5. Store any information (parameters) that came with the event in global memory
6. Send a message to a Clarion window that an event was received and the information is available in global memory.
7. Process the event reading the global memory
8. When done, disconnect, or in COM terminology 'Unadvise'
9. Uninitialize COM

Last week I covered the first five steps. Now its time to send a Windows message to a Clarion window telling the Clarion window that an event was received. This event is set up in the call to `ConPtCl.Init()`:

```
ConPtCl.init(window, 609H, thread(), 'Thanksgiving', cwdsynccl)
```

The first three parameters are the Clarion window, Clarion event, and Clarion thread that should receive notification when an event comes in from the COM object. In the sample code, the one and only window on the one and only thread gets an event

609H when a connected COM object generates an event. As mentioned above, a (Windows API) message is sent from CWDsyncCl to the Clarion window. While you could just pick a random Windows message to send, for safety your best bet is to supply a unique string and use it in RegisterWindowMessage(), which in turn generates a unique Windows message for you. By using RegisterWindowMessage() you are sure to get a unique Windows message not used by Windows or any other application calling RegisterWindowMessage().

The code in CWDsync.Init() generates the Windows message and subclasses the target Clarion window. When the subclass code receives the unique message, it posts (using POST()) the Clarion message (609H) to the Clarion thread passed to ConPtCl.init(). While this may seem like a long way to go to get event information into a Clarion Accept loop, it is thread safe. This is important since the events may come in while another Clarion thread is active, and any run-time library calls from the COM event when the runtime library doesn't expect it will cause errors. Fortunately, the CWDsync class takes care of all the details. If you are not familiar with subclassing, the technique I used here is the one I described in an earlier article (http://www.clarionmag.com/cmag/v2/v2n3subclass.html).

Now for the fun part – processing the event (step 7). When event 609H arrives the only two requirements are to read the data CWDsync put into blockcl and free any memory allocated when variant parameters were copied. This code does just that:

```
! in accept loop
of 609H
  loop
    if ConPtCl.GetNextEvent(EventId, parametercount, |
       Address(parameterArray)) |
    then break.
    !display/process the event
    Message('Mail Received')
    !clean up
    ConPtCl.VariantFree()
  end
```

EventID is the dispID you found earlier using OLEView. The parameterArray is defined like this:

```
ParameterArray like(vartype),dim(15)
```

Vartype is defined in conpt.inc and represents a variant:

```
vartype group,type
thetype    ushort
wreserved1 ushort
```

```
wreserved2 ushort
wreserved3 ushort
data1      long
pad        long
  end
```

The main purpose of variants is to make VB run slowly! The first `ushort` contains an constant that specifies the data. Some common values are 3=long, 2=short, 8=Bstring. The actual equate values for the types are available on MSDN. Data1 contains the actual data if the data type is a short or long but for a `Bstring` it contains the address of the `Bstring`. Consider storing a byte in VB – it would take 16 bytes to store the byte since all data is stored as variants! In addition you would spend a lot of time and code forever checking the data type. It is no wonder VB has a reputation for being slow.

The upside of VB's use of variants as a native data type is that no conversions are necessary for COM – the COM types are VB native types. I'm very glad Clarion does not use native COM types and gets the speed. I can bear the burden of converting to and from COM types for COM work.

When the `variantFree` method above is called, it calls the API `variantClear()` function which clears the variant structure (group), and also frees any memory the variant may point to such as a `Bstring`. Unless you are very sure there are no parameters or the parameters do not require any clean up, it is a very good idea to call `ConPt.VariantFree()`. In those cases where there are no parameters or you are not interested in the parameters, you can pass 0 or omit the third parameter of `GetNextEvent()` to save yourself the trouble of declaring the parameter array.

When ever you are done, post(`EVENT:CloseWindow`) to end the accept loop. Before disconnecting from Outlook you need to first tell Outlook to stop calling the `CWDSyncCl` (step 8). To do this you call the unadvised() method on the `IconnectionPoint` interface obtained way back in step 3. Fortunately the `ConPtCl` tracks all that and can tell Outlook to stop sending events without any further help then tells `cwdsyncCl` to shut down by calling its `kill` method.

The only thing left to do at this point, is to release the `Iunknown` interface to Outlook obtained in step 2. That is done by calling `Iunk.Release()`. Perhaps

An SMTP example

Also in the downloadable code is an example of getting COM Events while using the Clarion OLE Control. The example uses a free SMTP control (www.ostrosoft.com). On an error the control creates an error event that passes a bstring along with that event. If you are wondering what would happen if the user closed the running instance of Outlook you connected to in step 2 while your program is still running. Actually nothing happens. Outlook disappears from the desktop but it still appears on the task list. Outlook doesn't close down because when you attach to Outlook, `GetActiveObject()` automatically increases the reference count on Outlook; this happens when you get the address of Outlooks interface. Until you release that interface by calling its release method, there is no danger of Outlook closing down before you want it to.

Although the sample code uses early binding to interact with Outlook, you could just as easily used the Clarion OLE control and still get events. `ConPtCl` has an alternative `Connect()` method that takes a window and OLE control Field equate label rather than the address of an interface and can use that. The code just uses the OLE control's `prop:object` to get the address of it's interface and then calls the other connect method using the address obtained with `prop:object`.

Whenever you need events, the steps to follow are:

1. Examine the object in ole view to get the `IID` and `dispid` for the interface and event(s) you want. Also make note of any parameters.
2. Create an instance of `CWDSyncCl` and supply a `takeEvent` method if you need to filter out some events the interface can supply or do any special parameter handling.
3. Create an instance of `ConPtCl`.
4. Decide what Clarion window on which thread will receive what Clarion event when an event arrives. Pick a unique string to allow generation of a window event. Put all this information in to `ConPtCl.Init()`.
5. Obtain a the address of an interface on the COM object supplying the events or get the window and ole control field equate and pass that information along with the `IID` of the event interface obtained in step 1 to `ConPtCl.connect()`
6. When the events start rolling in call `ConPtCl.cwdsynccl.blockcl`, to read the `dispids` that determine what event was passed and any parameters.

you look at that code, you will see the conversion from `Bstring` to Clarion string is done just before the call to `ConPtCl.VariantFree()`. When testing the code you can force an error by calling the connect method and leaving the server property blank. When all the properties are properly set, you will get a 'connect', 'send', and 'close connection' event as the email is sent. In that example, no special code was needed in `CWDSyncCl.takeevent()` so the `TakeEvent()` method was not derived and the `cWDSyncCl` was not passed as a parameter to the `connect()` method. The `ConPtCl` class then uses `New()` to create an instance of the class for you when it is not passed.

7. When done call `ConPtCl.unadvised()` to shut things down then release the com object producing the events.

That’s about it. When a COM object has only one interface that can produce events, the built-in Clarion event handling may be worth a try owing to it’s simplicity, but when that fails, the method I’ve described here will always let you get the events you need.

Download the source

Jim Kane was not born any where near a log cabin. In fact he was born in New York City. After attending college at New York University, he went on to dental school at Harvard University. Troubled by vast numbers of unpaid bills, he accepted a U.S. Air Force Scholarship for dental school, and after graduating served in the US Air Force. He is now retired from the Air Force and writing software for ProDoc Inc., developer of legal document automation systems. In his spare time, he runs a computer consulting service, Productive Software Solutions. He is married to the former Jane Callahan of Cando, North Dakota. Jim and Jane have two children, Thomas and Amy.

Reader Comments

Add a comment

Clarion Magazine

Home   COL Archives

Topics > Design & Development > User Interface

Write A Word Processor In Five Minutes

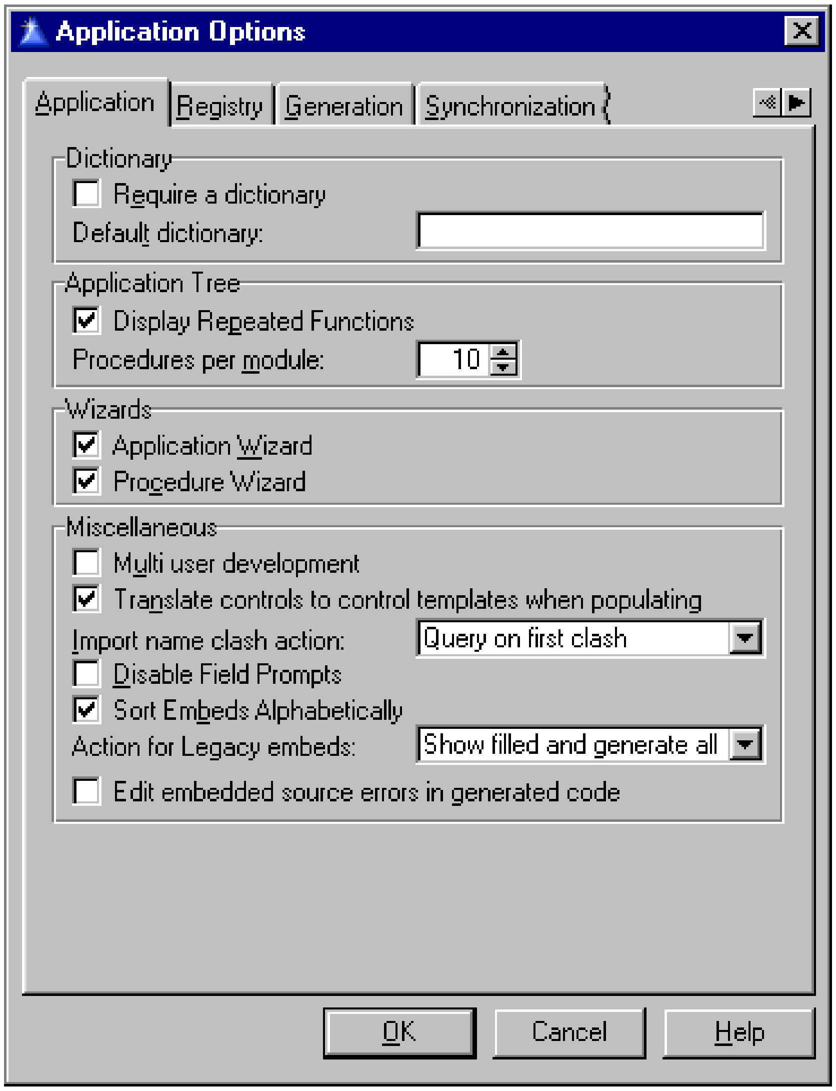
by Vince Du Beau

Published 2002-01-17

Recently I read a 30 page document that explained how to create a text editor in Delphi. Inspired, I decided to see how I could do this same task better and faster in my beloved Clarion. What I came up with was a very minimal word processing application that takes about 5 minutes to create. You will need Clarion 5.5 (or later) to follow along.

The application

If you haven't already done so, go into Setup -> Application Options and uncheck the "Require Dictionary" option, as shown at the top of Figure 1. You won't need any tables for this application.



Dictionary
- [ ] Require a dictionary
- Default dictionary:

Application Tree
- [✓] Display Repeated Functions
- Procedures per module: 10

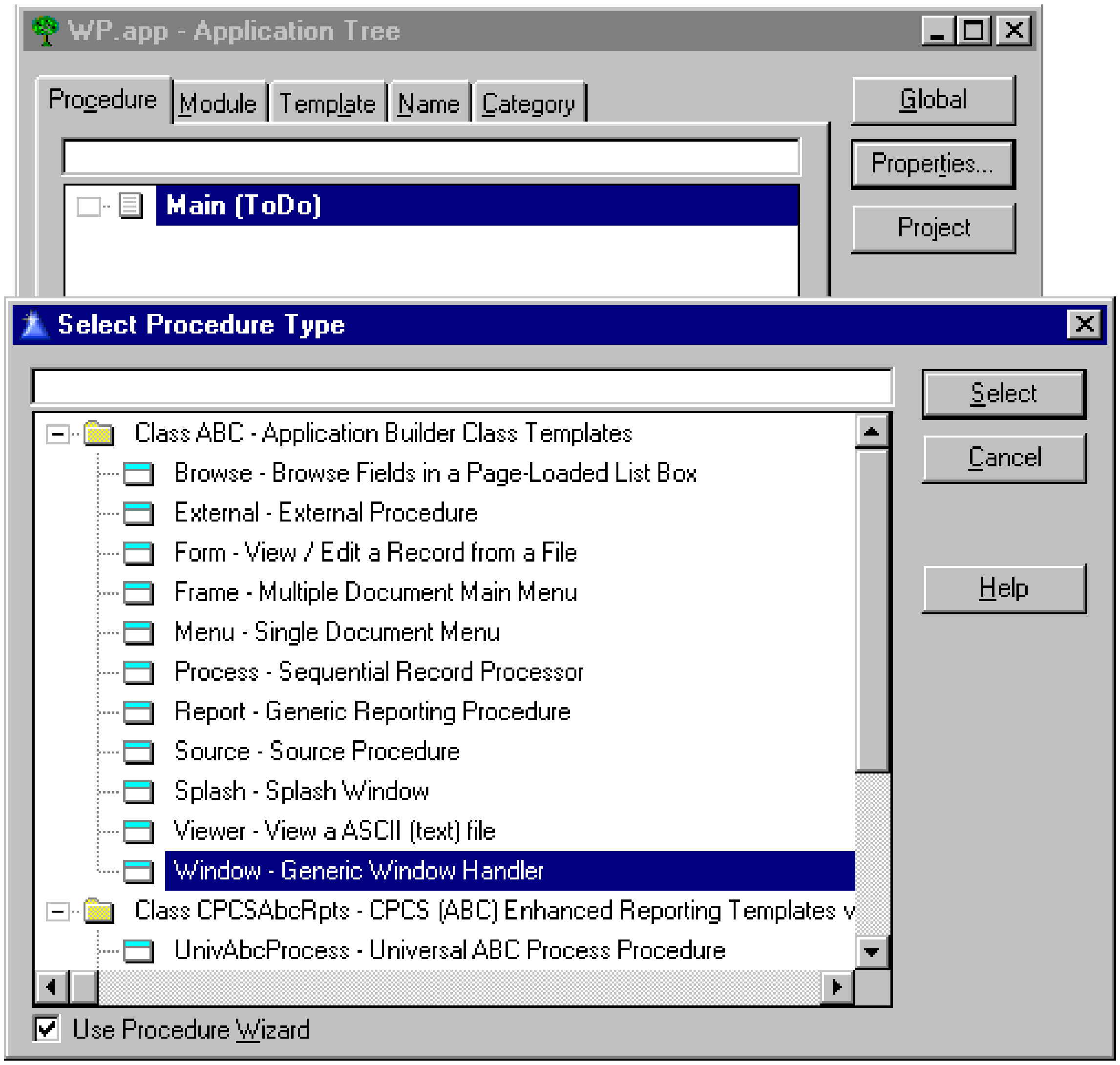
Wizards
- [✓] Application Wizard
- [✓] Procedure Wizard

Miscellaneous
- [ ] Multi user development
- [✓] Translate controls to control templates when populating
- Import name clash action: Query on first clash
- [ ] Disable Field Prompts
- [✓] Sort Embeds Alphabetically
- Action for Legacy embeds: Show filled and generate all
- [ ] Edit embedded source errors in generated code

Figure 1. Turning off the dictionary requirement in Application Options

Create a new application in Clarion in whatever directory you wish. Call it WP.app. Make sure that the "Use Quick Start" option is unchecked, and click on OK. In the application properties, uncheck the "Application Wizard" and click OK again. You're going to create the application's only procedure manually.

You should now be in the application's procedure tree view. Double click on the Main(ToDo) procedure and select Window – Generic Window Handler for the procedure type.



Class ABC - Application Builder Class Templates
- Browse - Browse Fields in a Page-Loaded List Box
- External - External Procedure
- Form - View / Edit a Record from a File
- Frame - Multiple Document Main Menu
- Menu - Single Document Menu
- Process - Sequential Record Processor
- Report - Generic Reporting Procedure
- Source - Source Procedure
- Splash - Splash Window
- Viewer - View a ASCII (text) file
- Window - Generic Window Handler

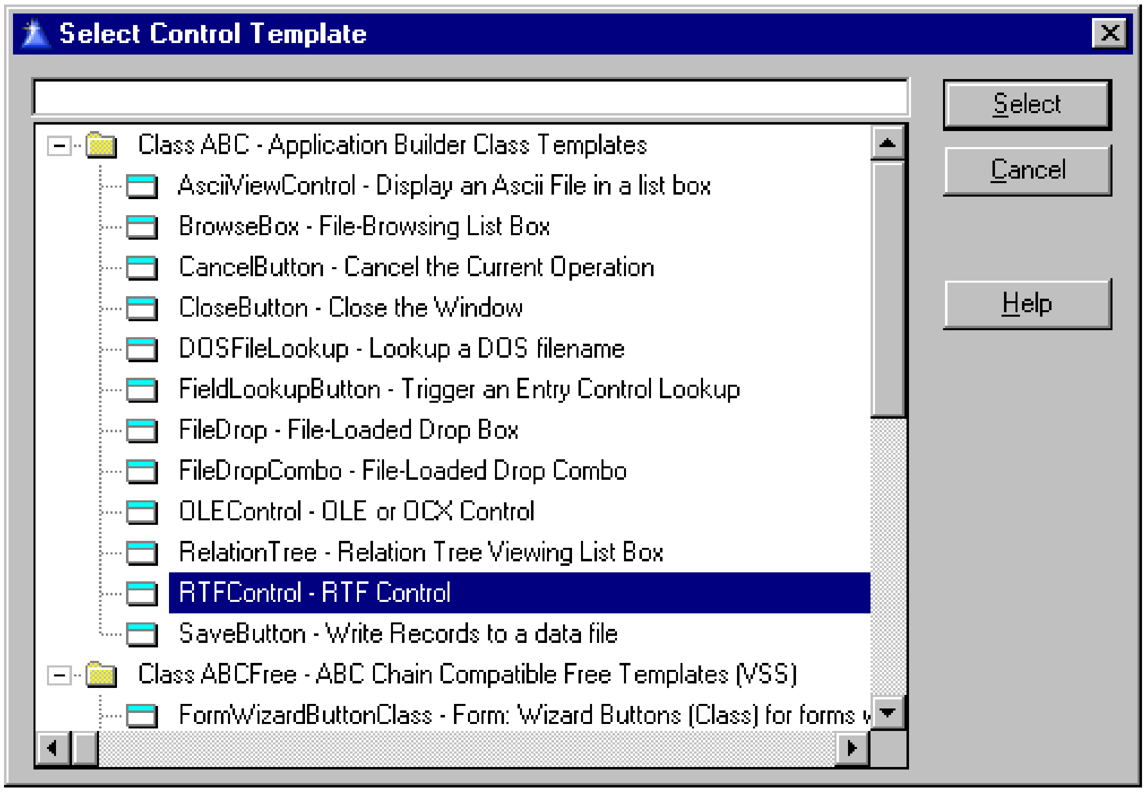
Class CPCSAbcRpts - CPCS (ABC) Enhanced Reporting Templates v
- UnivAbcProcess - Universal ABC Process Procedure

[✓] Use Procedure Wizard

Figure 2. Selecting a generic Window procedure template

Click on the "Window" button and select window as the structure. Bring up the properties box for the window. Under General Tab change the Text to "Word Processor" and change the border to resizable. Under the Extra Tab check the Immediate, Status Bar, and Maximize Box options. Click OK.

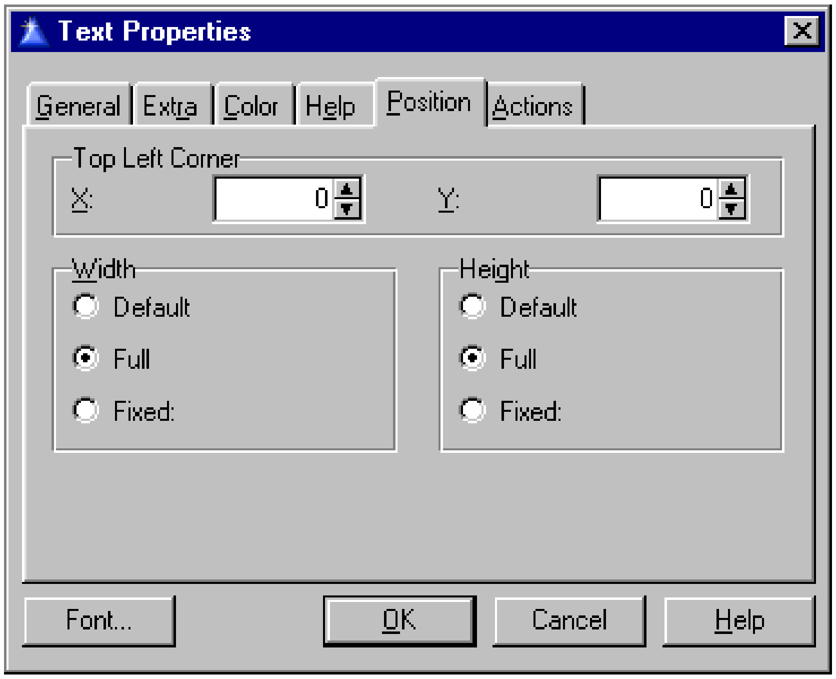
From the control toolbox, select the Control Template icon and then from the control list, choose the RTF Control template.



Class ABC - Application Builder Class Templates
AsciiViewControl - Display an Ascii File in a list box
BrowseBox - File-Browsing List Box
CancelButton - Cancel the Current Operation
CloseButton - Close the Window
DOSFileLookup - Lookup a DOS filename
FieldLookupButton - Trigger an Entry Control Lookup
FileDrop - File-Loaded Drop Box
FileDropCombo - File-Loaded Drop Combo
OLEControl - OLE or OCX Control
RelationTree - Relation Tree Viewing List Box
RTFControl - RTF Control
SaveButton - Write Records to a data file
Class ABCFree - ABC Chain Compatible Free Templates (VSS)
FormWizardButtonClass - Form: Wizard Buttons (Class) for forms v

Figure 3. Selecting the RTF control

Place the control on the window and bring up the property box. On the Position tab, set the top left corner X and Y positions to zero, and the width and height to Full. Click OK, save the window and exit to the procedure properties.



Top Left Corner
X: 0
Y: 0

Width
Default
[✓ Full]
Fixed:

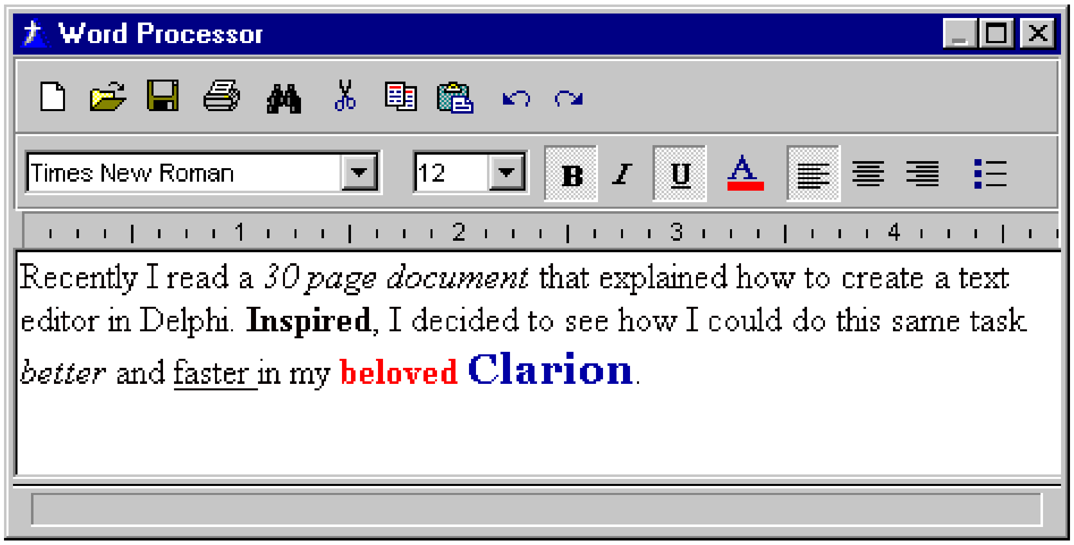
Height
Default
[✓ Full]
Fixed:

Figure 4. Setting the RTF control's position

Click on the "Extensions" button for the procedure and then the "Insert" button.

Select the Window Resize extension. Click the OK button and OK again to save the window.

From the Project Menu, click Run. The project will compile and run your new word processor with complete font control, cut and paste, save, and undo/redo.



Recently I read a *30 page document* that explained how to create a text editor in Delphi. **Inspired**, I decided to see how I could do this same task *better* and faster in my **beloved** **Clarion**.

Figure 5. The word processor in action

Summary

To create a complete word processor, you would need to add menus and a lot of other niceties (using the code templates that accompany the RTF control). But the next time someone wants to see what Clarion can do, whip out your laptop and build the word processor in five minutes or less.

Download the source

Vince Du Beau is an independent consultant working in New Jersey. He has been using Clarion since 1989. His company, Plover Development Group, Inc., does AS/400 consulting and custom PC development with Clarion.

Reader Comments

Add a comment

Excellent, just excellent.
excellent. put the text into a memo field or blob and you...
This was the most impressive example of Clarion's...
I was thinking, "please continue with a print button", as I...
There is a print button there although it doesn't...

Clarion Magazine

For marketing your Applications
and Developer Accessories or to
purchase other 3rd Party Tools . . .
Developer PLUS

Interfacing With C++ Part 1

by Gordon Smith

Published 2002-01-18

This article demonstrates how Clarion interfaces and C++ abstract base classes can be freely interchanged between Clarion, TopSpeed C++ and Microsoft Visual C++. Not only does this provide a convenient conduit for mixed language development, it also allows objects to be shared across languages. In practice it is possible to build a very powerful C++ library by simply wrapping existing or third party code within a Clarion compatible interface.

Interfaces

Interfaces are the key to using C++ abstract base classes in Clarion. But what are interfaces? The Clarion Help system describes them this way:

> "INTERFACE: A collection of methods to be used by the class that implements the interface."
>
> "An INTERFACE is a structure, which contains the methods (PROCEDUREs) that define the behavior to be implemented by a CLASS. It cannot contain any property declarations. All methods defined within the INTERFACE are implicitly virtual. A period or the END statement must terminate an INTERFACE structure."

That's the Clarion Help perspective. I prefer to describe an interface as a contract: it is a public declaration that class x supports contract y, thus any procedure or method that is willing to accept contract y can work with class x without knowing anything specific about class x.

Interfaces represent a great level of abstraction, which is demonstrated by the

`WindowManager.addItem` method. This method will accept as a parameter any class that has implemented a `WindowComponent` interface. In other words anyone who has developed a class which displays stuff on a window (calendar etc.) and who wants to make this class work within a `WindowManager`, simply needs to implement the `WindowComponent` interface for that class, thus fulfilling the contract.

The interface definition simply sets out the methods that any class which implements the interface must have. That's the contract: the interface guarantees the class will have certain methods. In the case of the `WindowComponent` interface, the `WindowManager` knows that any class passed to its `addItem` method will have `Kill`, `Reset`, `ResetRequired`, `SetAlerts`, `TakeEvent`, `Update`, and `UpdateWindow` methods which it can call. The obvious benefit is that the window manager needs no other knowledge about the visual component class.

> **NOTE**: Without interfaces the same result could be achieved by using a `WindowComponent` base class. Unfortunately this becomes very limiting in languages which do not have multiple inheritance (Clarion, Java, C#). This goes someway towards explaining the current trend in modern languages to only have single inheritance and to support interfaces.

This example also highlights the correct flow for interface design; in general people learning about OOP and interfaces make the mistake of writing a class and then trying to come up with an applicable interface for that class. That's the wrong way to approach it. It is better to design the interface from the receivers point of view (in the above case the receiver is the `WindowManager`).

Ok, enough preamble. It isn't this articles aim to teach you all about interfaces (for more information see the Interfaces topic), but to demonstrate how to mix and match Clarion interfaces with C++ "interfaces."

Mix and match

Now all you C++ people will be shouting that C++ doesn't have interfaces. This is true, kind of. C++ does however have multiple inheritance, and it has the ability to declare abstract base classes.

An abstract base class is a base class that contains pure virtual methods. These are methods declared in such a manner that they *must* be implemented in a derived class, but need not be implemented in the base class. Does that sound familiar?

In fact, if you declare a Clarion interface and a C++ abstract base class with identical methods, they become interchangeable. That is, it is possible to pass a Clarion-implemented interface into a C++ procedure and pass a C++ class (which has been derived from the matching abstract base class) into a Clarion procedure. Since both have stated that they support the named interface, they are equivalent and interchangeable.

Example 1: Clarion implemented interface, passed to a TopSpeed C++ procedure:

For this example, I've defined the following trivial interface called `TwoSignedInterface`. This interface is a contract, which states that the implementing class contains two signed variables. Here is the Clarion definition of the interface:

```
TwoSignedInterface  interface
getSignedA            procedure, signed
getSignedB            procedure, signed
                    end
```

The C++ equivalent looks like this:

```
class TwoSignedInterface
{
public:
  virtual signed getSignedA() = 0;
  virtual signed getSignedB() = 0;
};
```

What makes this an abstract base class is simply the use of virtual methods, which have `= 0` on the end.

Now on the Clarion side the implementing class looks like this:

```
TestClass class, implements(TwoSignedInterface), type
A           signed
B           signed
          end
  ...
TestClass.TwoSignedInterface.getSignedA procedure()

  code
  return self.A

TestClass.TwoSignedInterface.getSignedB procedure()

  code
  return self.B
```

And on the C++ side the receiving procedure, which takes a `TwoSignedInterface`, is as follows:

```
extern "C" signed cppAddTwoSigned(TwoSignedInterface &tsi)
{
  return tsi.getSignedA() + tsi.getSignedB();
}
```

The `extern` means that the procedure has external linkage, i.e. is available outside the `examcpp1.cpp` module; the `C` is used to tell the compiler/linker to use the standard C naming convention, as opposed to the C++ naming convention. This naming convention is *not* to be confused with the Clarion C attribute for procedures - in other words it changes how the procedure name is mangled.

Finally the Clarion declaration of the C++ procedure looks like this:

```
cppAddTwoSigned procedure(*TwoSignedInterface tsi), ↵
  signed, name('_cppAddTwoSigned')
```

If I had not used the C naming convention the declaration would have looked like this:

```
cppAddTwoSignedB procedure(*TwoSignedInterface tsi), ↵
  signed, name('_cppAddTwoSignedB@FR18TwoSignedInterface')
```

Not very nice! Also these naming conventions can change between compilers. A quick look at the Visual C++ example will show this. What is worse is, in VC++ the default mangled name is not compatible with Clarion LIB files, making it impossible to statically link to it. In general people writing API libraries will stick to the C naming convention as it allows the library to be used with multiple languages (and this includes people mixing C with C++).

Where is PASCAL, RAW?

Anyone familiar with API calls will be used to seeing `PASCAL,RAW` at the end of API function declarations. These attributes alter the calling convention (not to be confused with the naming convention I talked about earlier, again do *not* confuse the Clarion `C` attribute with C's `extern "C"`). `PASCAL` and `C` specify that parameters are passed on the stack; while `C` pushes the parameters left to right, `PASCAL` pushes them right to left. The `RAW` attribute tells Clarion not to push the length of any string / group (which it would by default).

NOTE: If you declare `const *cstring` instead of `*cstring` and do not have the `RAW` attribute the length is *not* pushed onto the stack.

As this first example only uses the TopSpeed compilers, which all share a standard calling convention (register based), there is no need to specify a different convention.

Example 2: TopSpeed C++ implemented interface, passed to a Clarion procedure:

Example 2 is the exact opposite to example 1. All classes are implemented in C++ and the addition is done in a Clarion procedure. Here is the Clarion definition of the interface:

```
TwoSignedInterface  interface
getSignedA            procedure, signed
getSignedB            procedure, signed
                    end
```

The C++ equivalent looks like this:

```
class TwoSignedInterface
{
public:
  virtual signed getSignedA() = 0;
  virtual signed getSignedB() = 0;
};
```

Both are unchanged from example 1, as we would expect. Now on the C++ side the implementing class looks like this:

```
class TestClass : public TwoSignedInterface
{
public:
  virtual signed getSignedA();
  virtual signed getSignedB();

  signed A;
  signed B;
};
...
signed TestClass::getSignedA()
{
  return A;
}

signed TestClass::getSignedB()
{
  return B;
}
```

And on the Clarion side the receiving procedure, which takes a `TwoSignedInterface`, is as follows:

```
clwAddTwoSigned procedure(*TwoSignedInterface tsi)

   code
   return tsi.getSignedA() + tsi.getSignedB()
```

Nothing remarkable here, it is just the exact opposite to example 1; the key point is that the two interfaces remain identical for both examples. For symmetry I even programmed the main entry point in C++ (something I haven't tried before!).

Next week I'll describe how to wrap a C++ standard template for use within Clarion.

Download the source

Prior to joining TopSpeed Development Centre, Gordon Smith worked for an Irish company developing software for multi-national pharmaceutical companies. He was also a member of the 3rd party accessories program (Compile Manager 2) and developed the Clarion Class Browser.

Reader Comments

Add a comment

Gordon, Excellent article! I will be re-reading this...
At last - I cannot describe how pleased I am to come across...
Very cool stuff! Thanks...

Clarion Magazine

Interfacing With C++ Part 2

by Gordon Smith

Published 2002-01-21

This is part two of an article that demonstrates how Clarion interfaces and C++ abstract base classes can be freely interchanged between Clarion, TopSpeed C++ and Microsoft Visual C++. Not only does this provide a convenient conduit for mixed language development, it also allows objects to be shared across languages. In practice it is possible to build a very powerful C++ library by simply wrapping existing or third party code within a Clarion compatible interface.

Last week I showed how to pass a Clarion interface to a TopSpeed C++ procedure, and vice versa. Now its time to do something practical with this knowledge.

Example 3: MS Visual C++ implemented interface, VC++ creator procedure and VC++ destructor procedure:

This example (hopefully) uses a somewhat real world example. It wraps the C++ standard template library's (STL) Output File Stream `ofstream` for use within Clarion. This STL class provides a simple way to output text to a file on disk; one reason it might be useful within Clarion is the ability to have several open files at once (without having to declare a separate file structure for each instance, as you would normally have to do). The VC++ project is called `example3`; it has been tweaked it to generate a DLL called `msvc_example3.dll`; I also created a Clarion compatible LIB file (called `msvc_ex3.lib`) using Clarions libmaker utility.

One of the great things about using interfaces to call between DLLs is that an interface has no effect on the exported list of procedures, so chopping and changing the interface will not require running the DLL through libmaker!

Here’s the Clarion code:

```
...
newOutputFile procedure(const *cstring filePath),|
  *IOutputFile, pascal, raw, name('_newOutputFile@4')
disposeOutputFile procedure(*IOutputFile f), |
  pascal, raw, name('_disposeOutputFile@4')
...

IOutputFile                 interface, com
appendText                    procedure(const *cstring text)
appendLine                    procedure(const *cstring text)
                            end
```

This is the Clarion equivalent of the normally published header file. There are two procedures and one interface: the procedures are used to create/destroy the instance of the class that implements the interface (see how the name mangling is different to the TopSpeed C++ compiler). Check out the `COM` attribute on the interface, before getting too excited (and before you start installing all your ActiveX controls); it is merely a shorthand convenience for the following:

```
IOutputFile       interface
appendText          procedure(const *cstring text), pascal
appendLine          procedure(const *cstring text), pascal
                  end
```

What is a bit nasty is that it doesn’t automatically include the `RAW` attribute, but in this example they are all `CONST` so it is ok. Next is the actual example:

```
tstA &IOutputFile, auto
cstrPathA cstring('c:\testA.txt')
cstrText1a cstring('Gordon')
cstrText1b cstring('Smith')

  code
  tstA &= newOutputFile(cstrPathA)

  tstA.appendText(cstrText1a)
  tstA.appendText(cstrSpace)

  disposeOutputFile(tstA)
```

Nothing special here, it just creates an instance of the interface, uses it and disposes it.

Here is the matching VC++ interface for IOutputFile:

```
class IOutputFile
{
public:
    virtual void __stdcall appendText(const char *text) = 0;
    virtual void __stdcall appendLine(const char *line) = 0;
};
```

The only difference from the TopSpeed example is the use of `__stdcall`; this tells the VC++ compiler to use the following calling conventions:

- Parameters are passed on stack, right to left (corresponding to the Clarion `PASCAL` attribute).
- Name-decoration convention: An underscore (_) is prefixed to the name. The name is followed by the at sign (@) followed by the number of bytes (in decimal) in the argument list. Therefore, the function declared as int func(int a, double b) is decorated as follows: `_func@12`

It isn't an accident that the VC++ convention matches the Clarion side, because this calling convention is what VC++ uses to declare its windows API prototypes.

Next is the class that implements the interface. All the code is inline, which means it is included in the definition:

```
class OutputStream : public IOutputFile
{
public:
    OutputStream(const char * filePath)
    {
        ofs = new std::ofstream(filePath);
    }
    ~OutputStream()
    {
        delete(ofs);
    }
    virtual void __stdcall appendText(const char * text)
    {
        *ofs << text;
    }
    virtual void __stdcall appendLine(const char *line)
    {
        *ofs << line << "\r\n";
    }

private:
    std::ofstream * ofs;
};
```

This is straightforward stuff. There is a constructor/destructor that will create/destroy `ofs` (which is an instance of the STL (using namespace STD) Output File Stream). Next come the implemented interface methods - these do the actual work (one with a `<13,10>` or `\r\n` in C++).

Finally there are the new/dispose procedures:

```
extern "C" __declspec(dllexport) IOutputFile *
  __stdcall newOutputFile(const char * filePath)
{
    return new OutputStream(filePath);
}

extern "C" __declspec(dllexport) void
  __stdcall disposeOutputFile(IOutputFile * s)
{
    delete((OutputStream *)s);
    return;
}
```

The only new thing here is the `__declspec(dllexport)`, which forces VC++ to export the procedures, making it equivalent to adding a procedures mangled name to a Clarion exp file.

Note that the individual methods are resolved within the interface based on offset (not name). Any differences between your declarations will be fatal; all methods must have matching parameter lists, return types, calling conventions and be in the same sequence (they do not need to have matching names). In general this is fine, but there is one nasty gotcha with the MS VC++ compiler: it will not preserve the order of overloaded methods (I think it sorts them based on the number of parameters, but am not 100% sure). So if you want to use overloaded names on the Clarion side you will need some judicious naming on the VC++ side. For example, consider the following Clarion interface:

```
SampleInterface    interface, com
getText              procedure(unsigned pos)
getText              procedure(unsigned startPos, unsigned endPos)
getText              procedure
                   end
```

You would to prototype the interface like this on the VC++ side:

```
class SampleInterface
{
public:
    virtual void __stdcall getTextA(unsigned pos) = 0;
    virtual void __stdcall getTextB(unsigned startPos,
      unsigned endPos) = 0;
    virtual void __stdcall getTextC() = 0;
}
```

Summary

In practice, my core library contains a single VC++ DLL with all my commonly used wrappers; where possible I statically link the wrapped functionality to reduce the number of third party DLLs I need to ship. Examples of C++ implemented interfaces include the STL String, STL Input File Stream, STL Output File Stream (make sure you get the latest bug fixes for these as the ones shipped with MSVC 6.0 are out of date), an example of a Clarion implemented interface include a set of callback methods being passed to the open source Expat SAX XML parser (a SAX parser is one which runs through the XML, calling back to predefined procedures when it gets an open tag, close tag body text etc.).

Hopefully this article will enhance your experience of OOP programming by enabling the use of different languages. At the very least it may help people trying to call into a VC++ DLL from Clarion. Have fun!

Download the source

Prior to joining TopSpeed Development Centre, Gordon Smith worked for an Irish company developing software for multi-national pharmaceutical companies. He was also a member of the 3rd party accessories program (Compile Manager 2) and developed the Clarion Class Browser.

Reader Comments

Add a comment

Hi Gordon, re...'In general this is fine, but there is...

Clarion Magazine

First Field, Required Field

by Steven Parker

Published 2002-01-25

It's funny how things work out sometimes. I had just spent several hours updating code designed to prevent embedded code executing when the Cancel button was pressed. Then came the posting on one of the newsgroups: "I have embed code for some fields that I want to prevent from firing if the user presses cancel. How can I do this?"

While it was nice to be able to help out another developer, in all candor, it was nicer to know that I wasn't the only one having this problem (read: "having this sort of bizarre requirement").

The problem this developer and I had run into is more complicated than it first appears to be. It is not confined to canceling *and* it does not always happen when canceling. The better known variation is that embedded code executes when a field has not been completed or modified in any way.

Confused?

Welcome to what I called the "Required Field Look Up Blues" when I first wrote about it in "Beating Those Required Field Look Up Blues," Clarion Tech Journal, 6, 2, March/April 1994. Clearly this is not a new issue (in fact, that article was a follow-up to "The Double Loop Does … You," Clarion Tech Journal, 6, 1, January/February 1994).

The phenomenon of embedded code executing even though the field had not been touched is caused by the double loop (which I'll explain in a moment). The execution of code when canceling is not. Yet both are direct consequences of way

Clarion windows are constructed.

How a Clarion window operates

OOP and ABC notwithstanding, the basic logic of a Clarion window has not changed since Designer was introduced in Clarion Professional Developer (for DOS) in 1986.

A Clarion window is a loop functioning substantially as follows:

```
Accept
  Case Event()
    !check non-control specific events
  End
  Case Selected()
    !check Event:Selected for each control
  End
  Case Accepted()
    !check Event:Accepted for each control
  End
End
```

(The actual code, in ABC forms, is not quite this simple. However, the logic is.)

Non-control specific events that are checked at the top of the loop include such things as the timer, gain/lose focus, open or close window and the like. `Case Selected()` occurs when a control is selected (i.e., you tab onto it or click it) and `Case Accepted()` occurs when a control is accepted (key strokes entered and/or tabbed off of, a button clicked, etc.)

So, a Clarion window is basically an event checking loop (in fact, it always has been). Therefore, if there is code embedded in one of these events, when the appropriate Case evaluates True, it will execute whether you expected it to or not.

The double loop

An Ok button (usually in Form procedures) populated on a window adds a little extra to the typical window. The developer's problem with Forms is that, while the window may open with the first field selected, the user can move to any other field on the form and can do so at any time, in any of several ways. Consider a basic retail store inventory entry form:

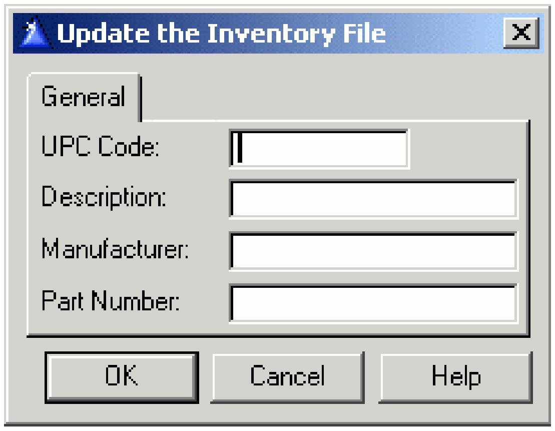
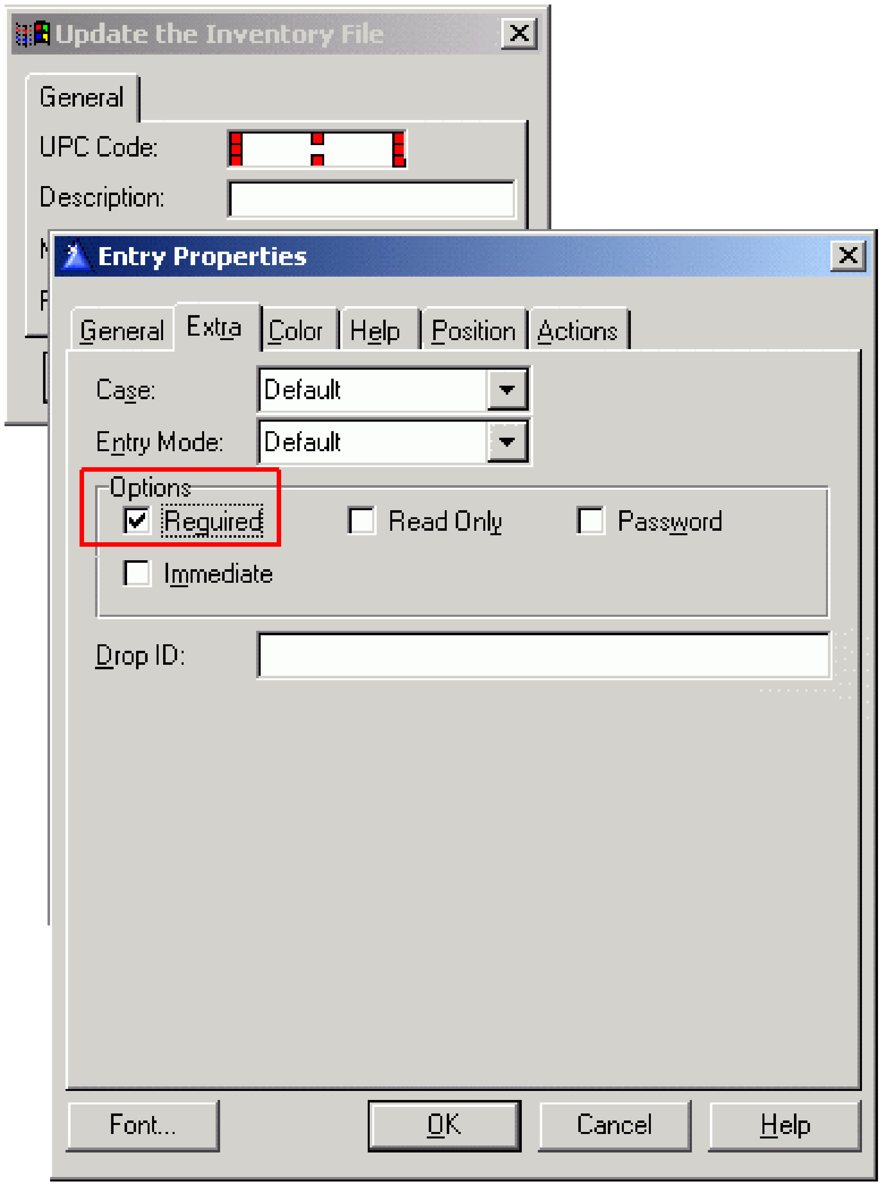


Figure 1. Basic inventory form

The UPC Code would typically be a required field. But a user could arbitrarily click on Description, make an entry and press "Ok."

If this form could be saved, if it were possible to save it, with UPC Code blank, just how is required field checking carried out?

Required field checking is a function of two settings. First, the field must have the Required attribute (see Figure 2).



Case: Default
Entry Mode: Default
Options
[✓ Required] Read Only Password
Immediate
Drop ID:

Figure 2. Required attribute

Usually this attribute is read from the dictionary, where the developer checks "Cannot be Zero or Blank" on the Validity Checks tab. But even if this attribute is not set in the dictionary, it can be set here, on the form. That is, by checking this box in a single procedure, a field can be made required for that procedure, even if it is not elsewhere. Likewise, by unchecking it here, a field can be made optional for a single procedure.

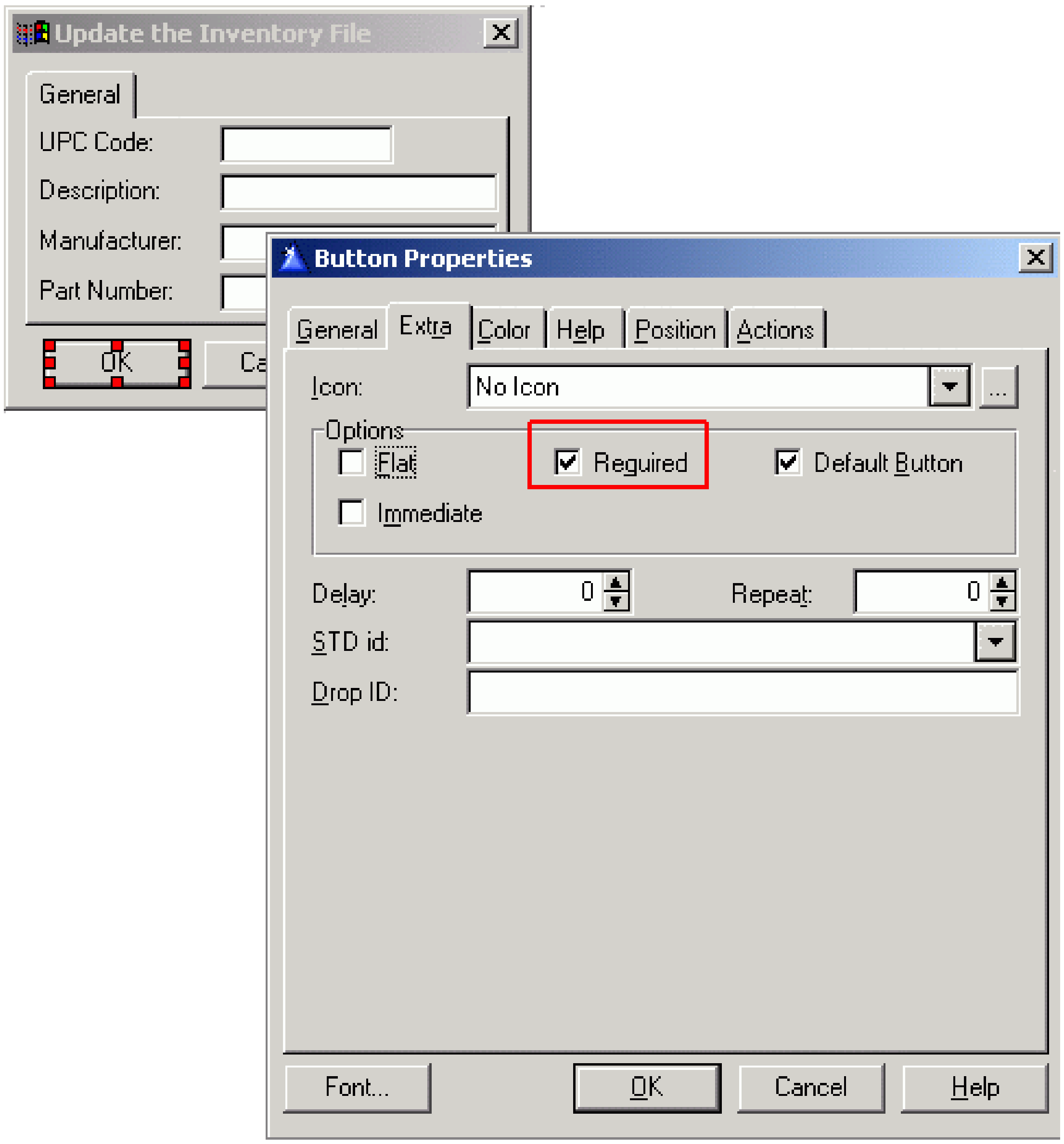
It is also possible to change whether or not a field is required in code. Any time after the window has been opened, add the following:

```
?INV:UPCCode{Prop:Req} = True
```

or, even

```
?INV:UPCCode{Prop:Req} = False
```

Second, the Ok button has to be told to check Required fields:



Icon: No Icon

Options
Flat [ ]
Required [✓]
Default Button [✓]
Immediate [ ]

Delay: 0
Repeat: 0
STD id:
Drop ID:

Figure 3. Turn on Required Fields checking

Again, all this is normally picked up from the dictionary, at least when you are using a wizard to initially create the form. But when you are creating a form without a wizard, it will *not* be checked. If required fields left blank are not being caught, this check box is likely not on.

What turning on the Ok button's Required option does is to cause the templates to add the `Select` statement (notice the absence of parameters) to the button's actions. Again, more complicated code is generated by the ABC templates but the logic is the same.

The on-line help states "SELECT with no parameters initiates `AcceptAll` mode (also called non-stop mode)."

> [AcceptAll] is a field edit mode in which each control in the window is processed by generating EVENT:Accepted for each. This allows data entry validation code to execute for all controls, including those that the user has not touched. AcceptAll mode immediately terminates when any of the following conditions is met:
>
> ```
> SELECT(?)
> Window{PROP:AcceptAll} = 0
> ```
>
> A REQ control is left blank or zero.

Read that again because *that* is how required field checking is enforced.

This means that every embed is executed when the window is in non-stop mode, explaining why embedded code may be executed twice.

If I embed:

```
INV:UnitPrice = (INV:Cost * INV:Markup) / INV:PackQty
```

double execution doesn't make a big difference. It may cost a few clock ticks but the data isn't going to be changed in unexpected ways.

But suppose I conditionally hide/unhide tabs. Then many cycles are going to be wasted. Or suppose I want to compute costs and margins only once (recalculating *can* overwrite a manual change) or I increment a value or counter (it would increment twice or it could increment simply by looking at the form and pressing Ok). In these cases, checking for non-stop mode allows me to short circuit the code:

```
If ~0{Prop:AcceptAll}
  DO CalculateCost
End
```

or

```
If ~FormWindow{Prop:AcceptAll}
  DO RetainPriceorMargin
End
```

In this way, I ensure that the routines are called only when the field has actually been changed and, then, only immediately after that change is made.

A more … interesting (ok, more frustrating) case occurs when attempting to intercept the "Creates Duplicate Key" message that so many of us … appreciate. You can do this by checking the `DUPLICATE` function, in the field's Accepted embed:

```
If ThisWindow.Request = InsertRecord
  Get(Inventory,0)
  If Duplicate(INV:UPCKey)
    Message('This Code already exists in the inventory.'|
      ,'Duplicate UPC',ICON:Hand)
    Select(?)
    Cycle
  End
End
```

However, when I press Ok and go into non-stop mode, `DUPLICATE` will return `True` even though the value *is* unique (or, at least, it was the first time it was checked). Changing this to:

```
If ThisWindow.Request = InsertRecord |
    and ~0{Prop:AcceptAll}
  Get(Inventory,0)
  If Duplicate(INV:UPCKey)
    Message('This UPC already exists in the inventory.'|
       ,'Duplicate UPC',ICON:Hand)
    Select(?)
    Cycle
  End
End
```

prevents the check from happening a second time. In so doing, I also prevent a spurious "error" message.

The double loop, at first sight, may seem like a bad design decision but, in addition to the fact that it can be easily by-passed, it ensures that each embed executes at least once and that required fields are completed. The alternative would be to check each field, either required or doing a calculation, in the Ok button. Manually. Much worse, I think, than the double loop.

Canceling

Canceling a form does not initiate non-stop mode. So how can embedded code execute when the user cancels?

It can't.

But it does.

Sometimes.

If you run the demo app and select the "Vanilla" procedures, you will find that you can tab around to your heart's content. You can Cancel without problem and required fields left blank are caught when you press "Ok." All of this works exactly as you would expect.

If you select the inventory browse/form from the "Standard Mods" menu (which adds a lookup to manufacturer in the inventory form), you will find the same. All goes exactly as you would expect.

But is it what you want?

Of course it is just this ability to move all over the form, at will, that is the problem. A user can pass the uniquely identifying field, UPC Code in this case, without having to enter it. Ditto the lookup. It is not until the user tries to save the form that they are forced to make an entry in that field. (Indeed, the apparent ability to leave a required field empty was part of my motivation for writing "Beating Those Required Field Look Up Blues.")

Some believe that stopping the user and forcing entry of such important fields immediately is better than making them do so after they think they're done with the form. I am one who happens to believe this. (In this case, especially, as the form is designed to find the next available number if none is entered here. If I didn't enforce the entry immediately, I could end up with a wasted auto-number.)

To force tabbing off a field to behave as if an entry had been made (like the old DOS `GLO:ForceValidate` switch), many of us have learned to add:

```
?INV:UPCCode{Prop:Touched}
```

to the Selected embed for the field.

With this code embedded, it is no longer possible to tab off this field without making an entry (see "Standard Mods" | Manufacturers to see this in action).

`Prop:Touched` is logically equivalent to making an entry in a field. From the Help:

> When non-zero, indicates the data in the ENTRY, TEXT, SPIN, or COMBO control with input focus has been changed by the user since the last EVENT:Accepted. This is automatically reset to zero each time the control generates an EVENT:Accepted. Setting this property (in EVENT:Selected) allows you to ensure that EVENT:Accepted generates to force data validation code to execute, overriding Windows' standard behavior—[standard Windows behavior is that] simply pressing TAB to navigate to another control does not automatically generate EVENT:Accepted.

And *this* is where the problems start. If you bring up the manufacturer form to insert a new record, then reconsider and press Cancel, you will get the `Message()` I populated in the `Accepted` embed.

Pressing Ok on the message and canceling again will close the form. This sort of behavior is not acceptable. But what's a developer to do?

When a user presses Cancel, `~0{Prop:AcceptAll}` will not stop the code from executing (`{Prop:AcceptAll}` is False, so `~0{Prop:AcceptAll}` is True and the condition passes).

Testing `~(Self.Response = RequestCancelled)` is no more helpful. For, while the window *has* been cancelled, `Self.Response` is often zero in a Clarion window but `GlobalResponse` expects 1 and 2 as values.

In other words, you're checking for a value of 2. But `ThisWindow.Response` may contain 0 or 2. I found it is zero when tabbing through a field on `ChangeRecord` but 2 when action is `InsertRecord`, which is less than helpful. Furthermore, if you embed

```
If ~(Self.Response = RequestCancelled)
!code
```

in the required field's `Accepted` embed, it will not be checked when the Cancel button is pressed (see the demo app's "Standard Mods" Manufacturer form) and the embed will still execute. And, if no fields have been Accepted, I wouldn't expect it to.

Confused?

What *is* a developer to do?

It seems as if I can enforce field validation or I can have an acceptable Cancel but not both.

Back to DOS

In my DOS applications the required field was either the first field on the form or the first non-display field (for a short while, I entertained the hope that putting another field first would alleviate the problem). Every case I had was like this. That’s why I started calling it the "first field-required field" problem.

My solution to this make-an-entry-or-you-can’t-cancel problem was to add the Escape key as an alerted key for the first required field. Then, in that field’s `Selected` embed I placed either:

```
If KeyCode() = EscKey
  Do ProcedureReturn
End
```

or

```
If KeyCode() = EscKey
  Select(?Cancel)
  Press(EnterKey)
End
```

(The `ProcedureReturn Routine` was introduced fairly late in the CDD development cycle and that indicates just how old this solution really is.)

This technique works because the `ALRT()` attribute causes the nominated key to select and complete the field (see the DOS Language Reference, 7-37). That is,

```
If KeyCode() = specified key
```

is added to the `Case KeyCode()` check at the top of the Accept loop. This is the logical equivalent of adding it to the non-control specific event checking of a window described earlier.

It took a reminder from Jeff Slarve and Carl Barnes to get me back on track: `Alert` key processing *precedes* all other field processing.

My final answer

Unfortunately, the final answer is not quite so simple.

For the many cases, circumventing the effects of `{Prop:Touched}` can be achieved by `Alert`ing a key for the Cancel button, a key not likely to be used, as shown in Figure 4.

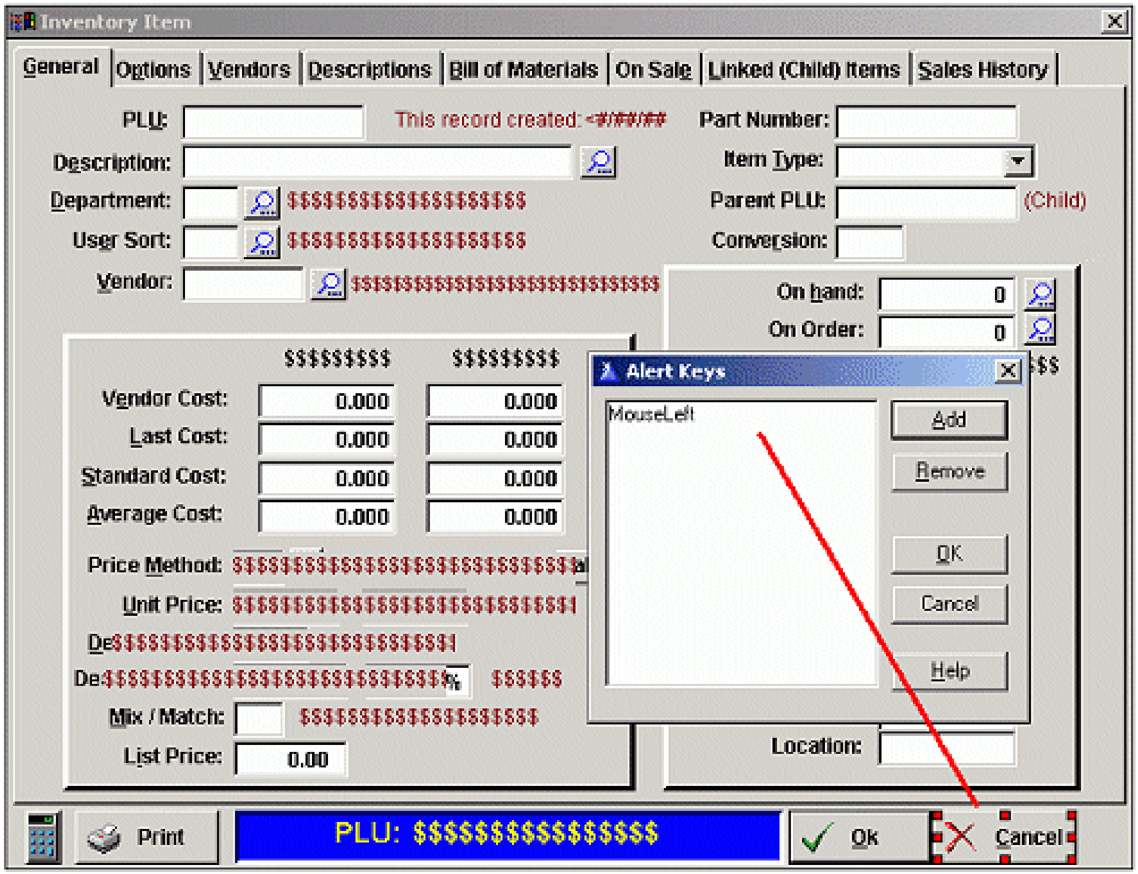


Figure 4. Alerting MouseLeft for the Cancel button

Because Alert keys are handled first, you can close the window in the `Event:AlertKey` embed (see Figure 5).

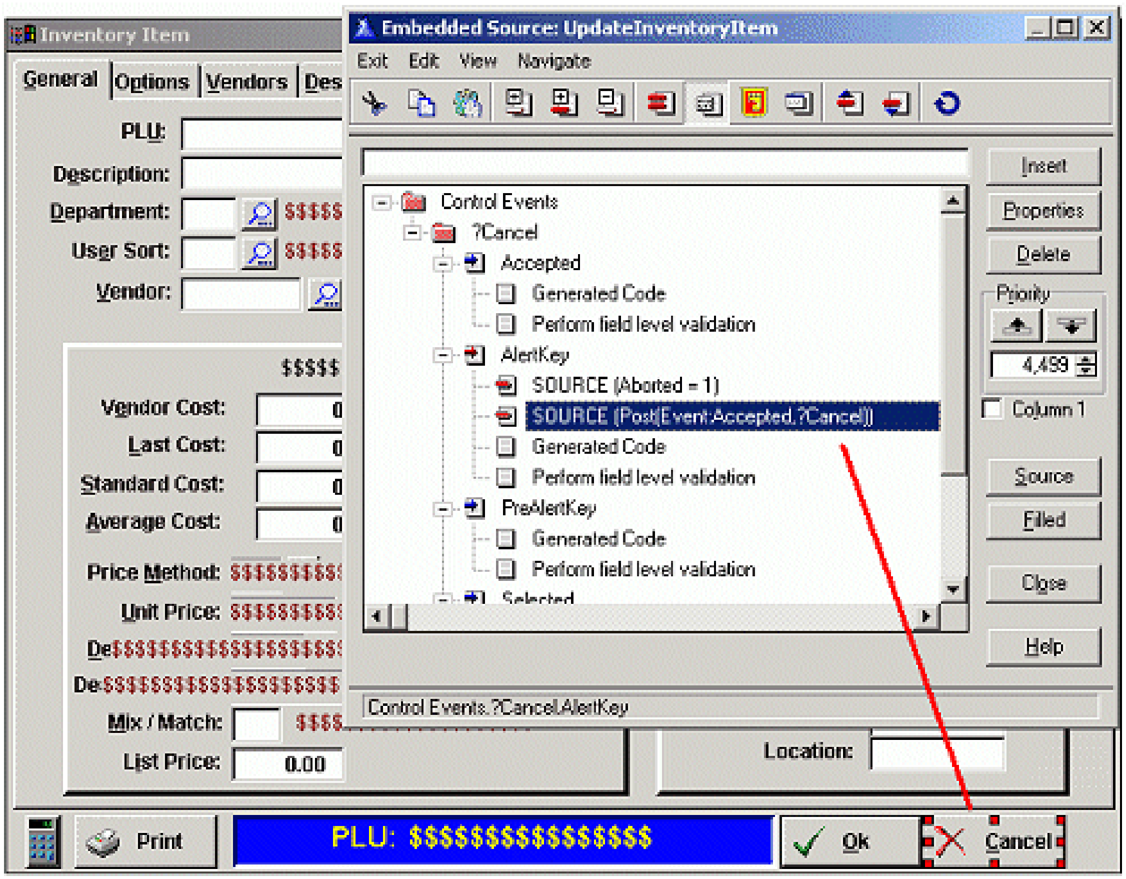


Figure 5. Forcing the window to close

However, embedded code in the "touched" field (with `{Prop:Touched}` set) will still execute. Because of `{Prop:Touched}`, per the Language Reference, the `Accepted` embed must execute and must do so before Cancel can post. This also explains why `ThisWindow.Response` is unreliable – the Cancel button has not had a chance to set the value.

As discussed above, I cannot successfully check `ThisWindow.Request`, so I created my own. I have local variable, `Aborted`, and I set it in the `AlertKey` embed (see Figure 5, above). Then I updated my embedded code to check for this variable:

```
If ThisWindow.Request = InsertRecord and |
   ~0{Prop:AcceptAll} and ~Aborted   !Check UPC Code
  If ~INV:UPC            !no UPC Code entered
    INV:UPC = GetNextUPC()
    Display(?INV:UPC)
    Select(?INV:UPC)
  Else                   !code entered, check for duplicate
    Get(Invtry,0)
    If Duplicate(INV:UPCKey)
      Message('This UPC already exists in the inventory.'|
        ,'Duplicate UPC',ICON:Hand)
      INV:UPC = GetNextUPC()
      Select(?)
      Cycle
    End
  End
  If FirstLoop
    FirstLoop = 0
    Select(?INV:Description)
  End
End
```

Et voila, the code does not execute when the user cancels.

Summary

It wouldn’t surprise me in the least to discover that my DOS solution, alerting the Escape key on the field with embedded code worked unchanged in Windows. Clarion after all is still Clarion and a Clarion screen and a Clarion window are remarkably similar.

If I’m coding in a `Loop`, many things suddenly get a lot easier – *if* I know or can figure out where I am in the `Loop`. Let’s see, reports are loops ("Completely Dynamic Report Orders and Breaks (Part 1)"), processes are loops and, now, windows are loops.

Am I seeing a pattern here?

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. A former SCCA competitor, he has been known to adjust other competitors' right side mirrors - while on the track (but only while accelerating). Steve has been writing on Clarion since 1993.

Reader Comments

Add a comment

The Ok button you refer to at times probably should be...
You are quite right, when using a Window template, the Save...

Getting A Handle On The System Tray

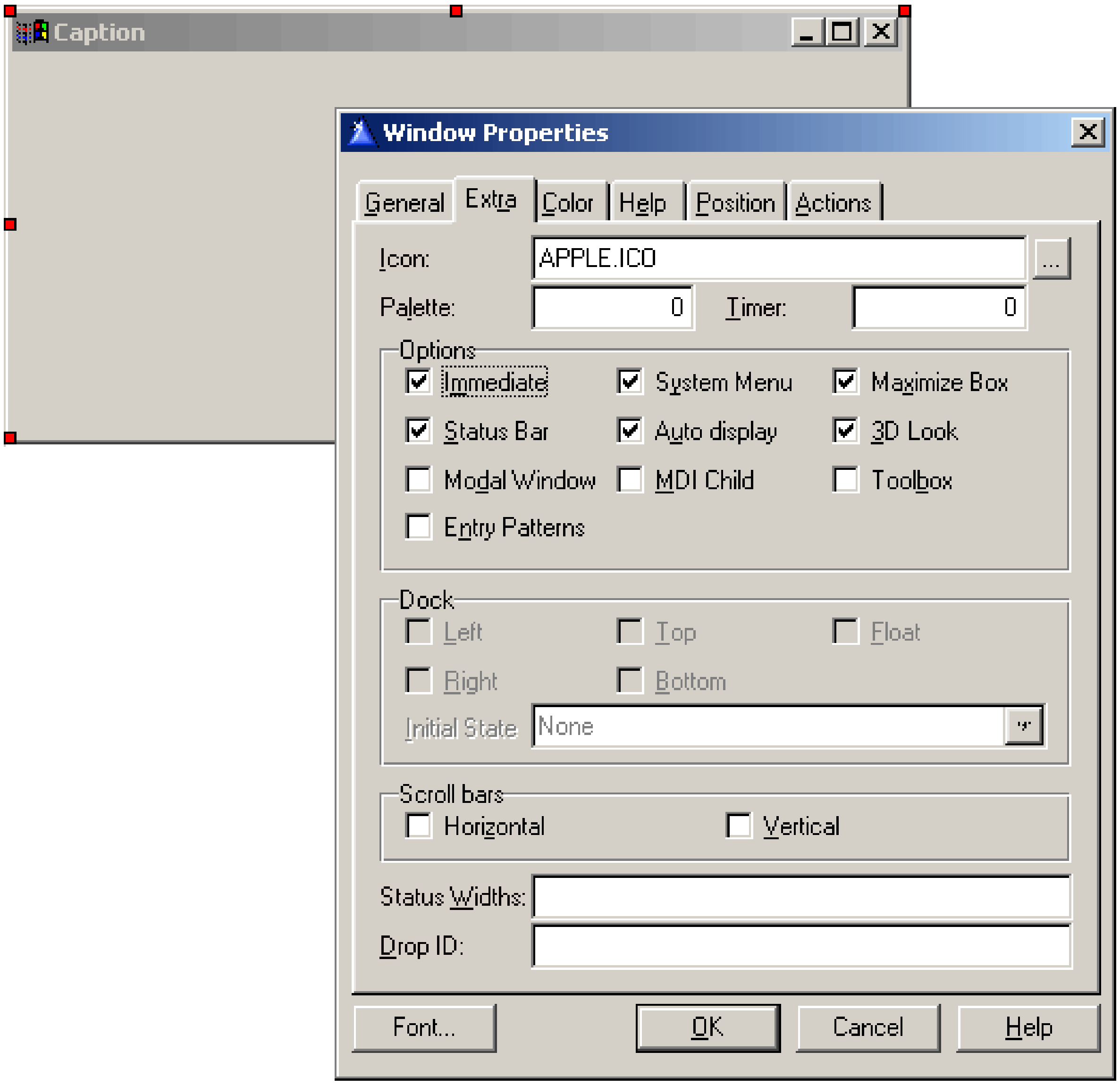
by James Cooke

Published 2002-01-30

One main benefit of Windows programming is being able to have multiple applications open at once. The problem with this is that if you have ten programs active at once, you have ten programs cluttering up your task bar, and ten icons to Alt-Tab through to land on your desired application. So – the system tray to the rescue! This article will cover the basic steps required for parking an app in the system tray, and responding to events on that icon.

Step 1: load the image

Create a new 32 bit application called `trayicon.app` with a single procedure called `MAIN` and define a window for it. Define `APPLE.ICO` as the icon for the window (this icon is supplied with the example application that accompanies this article).



Icon: APPLE.ICO

Palette: 0 Timer: 0

Options

✓ Immediate	✓ System Menu	✓ Maximize Box
✓ Status Bar	✓ Auto display	✓ 3D Look
☐ Modal Window	☐ MDI Child	☐ Toolbox
☐ Entry Patterns		

Dock

☐ Left	☐ Top	☐ Float
☐ Right	☐ Bottom	

Initial State: None

Scroll bars

☐ Horizontal ☐ Vertical

Status Widths:

Drop ID:

Figure 1. Defining the window and icon

Because you have added the icon to the window’s definition, Clarion adds the icon as a resource into its project and links it into the executable. This will allow you to deploy the executable on its own, without having to include a bunch of icons.

One thing to understand is that Clarion provides the programmer with a much simplified means of managing images. Windows does not deal directly with filenames - rather, it provides the programmer with an API function called *LoadImage()*, which preloads an image into memory and provides the programmer with a *handle* to that image. This handle is merely the memory address of the image, and you use it when referencing the image in your code.

Add this Windows API prototype in the global map:

```
MODULE('Whatever')
  LoadImage(Unsigned,*Cstring,Unsigned,Signed,Signed,|
    Unsigned),Unsigned,Pascal,Raw,Name('LoadImageA')
END
```

A note on the Windows API: adding a prototype to the global map has the effect of extending the Clarion language. The new language statement is provided by the Windows operating system; and as such is available to all Windows languages. You can find details on all Windows API functions at the Microsoft Developer Network web site (msdn.microsoft.com)

Define these local variables:

```
IconName     CSTRING(80)
HAppleIcon   ULONG  !Handle of icon returned by LoadImage()
```

Embed this code after opening the window:

```
IconName        =   'apple_ico'
HAppleIcon      =   LoadImage(SYSTEM{PROP:AppInstance},|
                    IconName,1,16,16,0)
```

This code loads the icon into system memory, and makes it available to your application via the `HAppleIcon` handle.

Step 2: Add the icon to the tray

Add the `Shell_NotifyIcon` prototype to the global map:

```
Shell_NotifyIcon(Unsigned,Signed),Long, Pascal,Name ('Shell_NotifyIconA')
```

Your API section should now look like this:

```
MODULE('Whatever')
    LoadImage(Unsigned,*Cstring,Unsigned,Signed,Signed,|
    Unsigned),Unsigned,Pascal,Raw,Name('LoadImageA')

    Shell_NotifyIcon(Unsigned,Signed),Long, Pascal,Name|
    ('Shell_NotifyIconA')
END
```

The `Shell_NotifyIcon` function expects two parameters. The first is an indicator of whether you are adding, changing or deleting an icon; the second is the address of a `GROUP` structure which contains a collection of important information about the icon, and a series of flags. These flags are defined as global constants.

Embed these constant definitions in the global section, after global includes:

```
NIF_MESSAGE      EQUATE(1)   !Flag used by Shell_NotifyIcon
NIF_ICON         EQUATE(2)   !Flag used by Shell_NotifyIcon
NIF_TIP          EQUATE(4)   !Flag used by Shell_NotifyIcon
NIM_ADD          EQUATE(0)   !Flag used by Shell_NotifyIcon
NIM_DELETE       EQUATE(2)   !Flag used by Shell_NotifyIcon
Event:NIM        EQUATE(440h)!User Defined callback event
```

Define the group by embedding these local data definitions into MAIN:

```
NotifyIconData      GROUP,PRE(NID)
cbSize                  ULONG
hWnd                    UNSIGNED
uID                     UNSIGNED
uFlags                  UNSIGNED
uCBmessage              UNSIGNED
hIcon                   UNSIGNED
ToolTip                 CSTRING(64)
                    END
ID_AppleIcon    EQUATE(100)
Err             LONG
```

Now prime the above group with the relevant values (after the `LoadImage` code):

```
NID:cbSize      =   SIZE(NotifyIconData)
!the number of bytes in this GROUP

NID:hWnd        =   Window{Prop:Handle}
!the handle of the process that uses this image

NID:uId         =   ID_AppleIcon
!The applications' ID for this icon. A contstant

NID:uFlags      =   NIF_ICON + NIF_MESSAGE + NIF_TIP
!Indicate how the icon is displayed

NID:uCBmessage  =   Event:NIM
!A Callback event. (Locally declared event that Windows
!                   uses to notify us of events outside
!                   the application)

NID:hIcon       =   HAppleIcon
!The handle of the Apple ICon

NID:ToolTip     =   'Apples are good for you.'
!The tooltip used for the icon
```

At this point, you are ready to add the icon to the system tray: Place two buttons on the window and set the text of the first to `Add` and the second to `Remove`.

Embed this code into the Add button:

```
Err = Shell_NotifyIcon(NIM_ADD,ADDRESS(NotifyIconData))
```

Embed this code into the Remove button:

```
Err = Shell_NotifyIcon(NIM_DELETE,ADDRESS(NotifyIconData))
```

That's it! Run the application and click the two buttons – when you click the `Add` button an apple will appear in the system tray, when you click the `Remove` button it will disappear. Hover the mouse over the apple – the tip should read, "Apples are good for you." Another thing: close down the application before removing the icon, and the apple stays there. That's because Windows has not had an explicit instruction to remove it. However, since one of the `NotifyIconData` members that passed it was the handle of the process, Windows is able to use that handle to check if the process is still running. Since the application is closed, the handle is invalid, and Windows deletes the image the moment the image fires any events. Move your mouse over the image, and you will see it disappears automatically; Windows responds to the `MouseOver` event, discovers the handle is invalid and destroys the image.

Step 3: Trapping tray icon events

It's well and good being able to display an icon; but the challenge is to render some sort of communication between Windows and the application. This is necessary because the application needs to respond to events that happen in the tray, which the Clarion APP has no access to. There is only one event that is of importance to this application: when the user right-clicks the icon, the application needs to be notified so that it can execute some relevant code, for example a popup menu with some options on it.

This is where `Event:NIM` becomes useful. Remember, before calling `Shell_NotifyIcon` the `GROUP` variable `NID:uCBmessage` was assigned the value contained in `Event:NIM`. This means that Windows will fire off the application's event `Event:NIM` when it needs to send the application a message.

In order to trap that `Event:Nim`, it will be necessary to use a *subclass procedure*. All a subclass procedure does is it allows a procedure to trap Windows events that Clarion does not trap automatically. It can then call other Clarion code or post an event to a window in your application.

Define the subclass procedure

The subclass procedure makes use of a single Windows API call. Add this prototype to the global map:

```
CallWindowProc(ulong,uLONG,UNSIGNED,UNSIGNED,uLONG),|
  ulong,PASCAL,NAME('CallWindowProcA')
```

Your API section should now look like this:

```
MODULE('Whatever')
    LoadImage(Unsigned,*Cstring,Unsigned,Signed,Signed,|
    Unsigned),Unsigned,Pascal,Raw,Name('LoadImageA')

    Shell_NotifyIcon(Unsigned,Signed),Long, Pascal,Name|
    ('Shell_NotifyIconA')

    CallWindowProc(ulong,uLONG,UNSIGNED,UNSIGNED,uLONG),|
    ulong,PASCAL,NAME('CallWindowProcA')
END
```

Define a new Source procedure called `MainSubClassFunc`

Set the prototype to: `(Unsigned,Unsigned,Unsigned,Long),Long,Pascal`

Set the parameters to: `(hWnd,wMsg,wParam,lParam)`

Embed the following in the code section:

```
Case wMsg
Of Event:NIM
   !check for tray icon window message
   Case Band(lParam, 0FFFFh)
   OF WM_RBUTTONUP
       Post(Event:NIM:MouseRight)
   End
   Return(0)
End
 Return(CallWindowProc(OrigWndProc,hWnd,wMsg,wParam,lParam))
 !process other window messages in
 !standard CallBack procedure
```

This code traps the external `Event:NIM` (fired by Windows) and posts an internal, user-defined event (`Event:NIM:MouseRight`) to the `MAIN` procedure. This "pass-the-parcel" idea is the essence of subclassing events, and can be used for many different applications.

In order for this code to compile, you will also need to define two more constants. Add the following constants to the Global Includes section:

```
WM_RBUTTONUP         EQUATE(205h)
                     !Windows' value for RightMouseUp
Event:NIM:MouseRight EQUATE(442h)
```

```
                          !User Defined event for this app
```

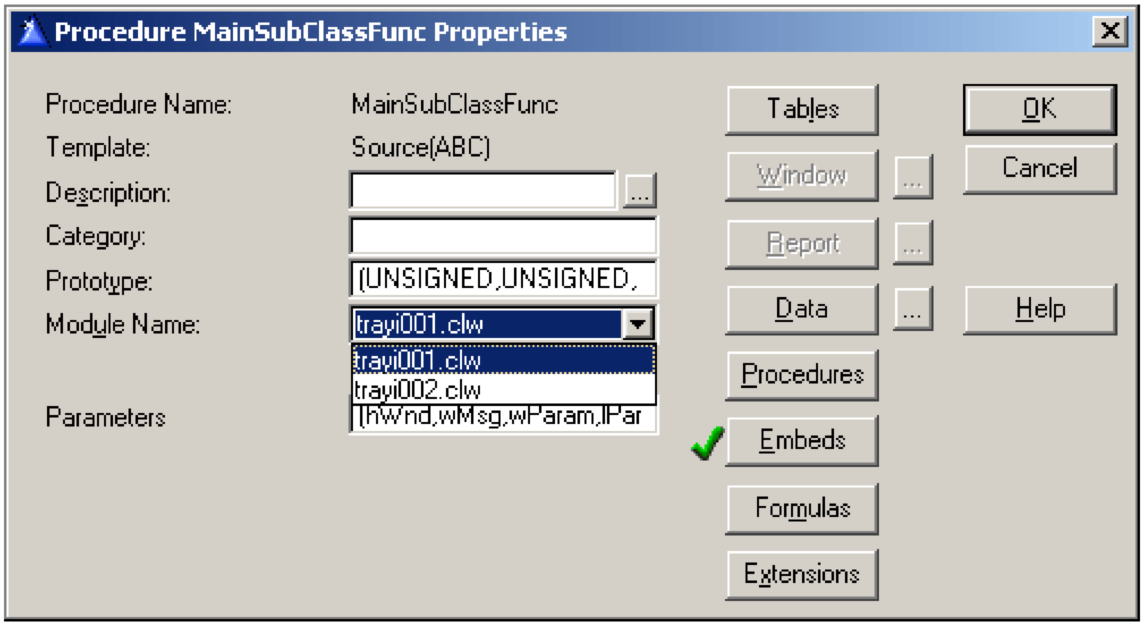
The Global Includes section should now look like this:

```
NIF_MESSAGE               EQUATE(1)   !Flag used by Shell_NotifyIcon
NIF_ICON                  EQUATE(2)   !Flag used by Shell_NotifyIcon
NIF_TIP                   EQUATE(4)   !Flag used by Shell_NotifyIcon
NIM_ADD                   EQUATE(0)   !Flag used by Shell_NotifyIcon
NIM_DELETE                EQUATE(2)   !Flag used by Shell_NotifyIcon
Event:NIM                 EQUATE(440h)!User Defined callback event

WM_RBUTTONUP              EQUATE(205h)
                          !Windows' value for RightMouseUp
Event:NIM:MouseRight EQUATE(442h)
                          !User Defined event for this app
```

Since it is necessary for the two procedures to share a common variable, and the use of global variables is taboo, you'll need to place both procedures in the same module. By declaring a module level variable you allow the two procedures to communicate easily without making that variable visible to the rest of the application.

Change the module of `MainSubClassFunc` to `trayi001.clw` (assuming that is the name of the main procedure). When Clarion Generates the code both procedures will be generated into `trayi001.clw`. This is illustrated in Figure 2.


Procedure MainSubClassFunc Properties

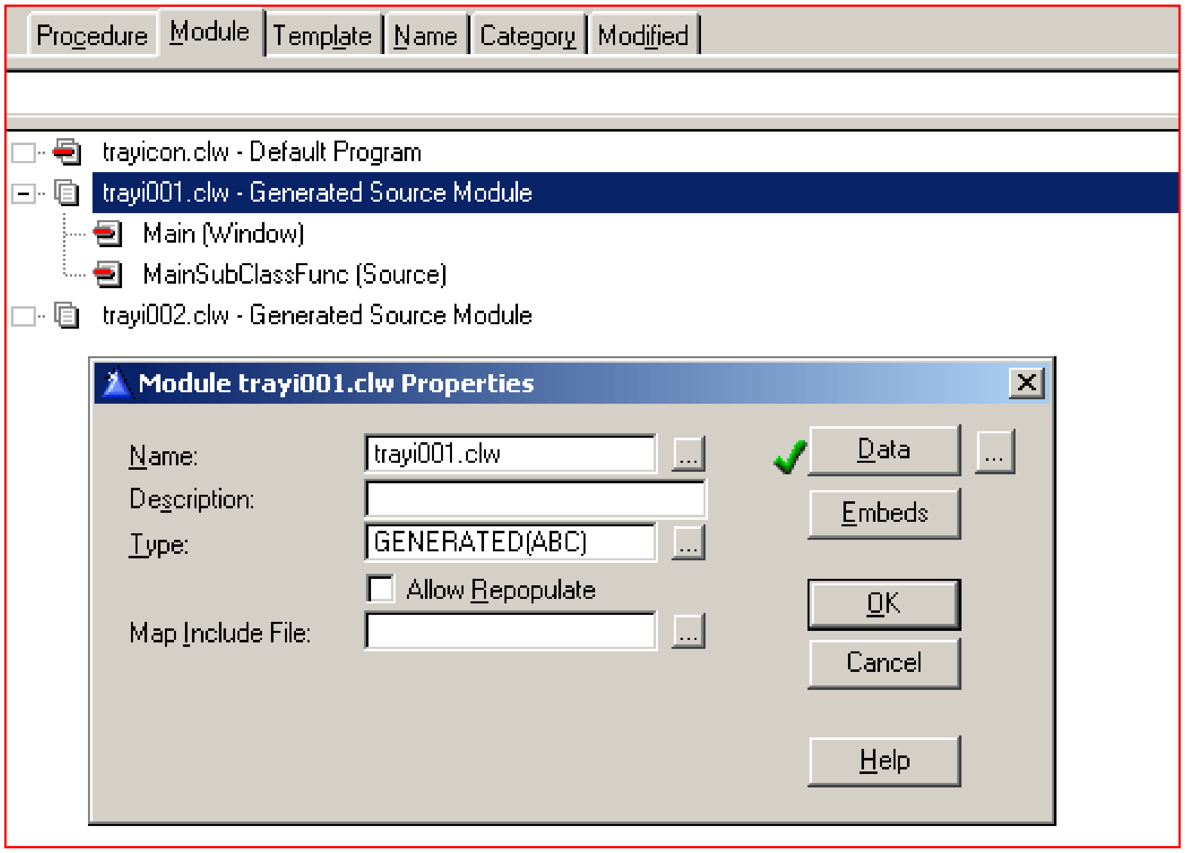
Procedure Name: MainSubClassFunc
Template: Source(ABC)
Description:
Category:
Prototype: (UNSIGNED,UNSIGNED,
Module Name: trayi001.clw
trayi001.clw
trayi002.clw
Parameters (hWnd,wMsg,wParam,lPar

Figure 2. Changing the procedure's module.

Click OK, then go to MODULE view in the Application tree. Delete `trayi002.clw` (it's just hanging around doing nothing) and add this Module level variable to the data section for trayi001.clw.

```
OrigWndProc LONG
```

Uncheck the Allow Repopulate checkbox to prevent the procedures from being reassigned to other modules later on. Click OK to save the settings and to return to the application tree.



Procedure | Module | Template | Name | Category | Modified

- trayicon.clw - Default Program
- trayi001.clw - Generated Source Module
  - Main (Window)
  - MainSubClassFunc (Source)
- trayi002.clw - Generated Source Module

Module trayi001.clw Properties

Name: trayi001.clw
Description:
Type: GENERATED(ABC)
Allow Repopulate
Map Include File:

Figure 3. Defining a module level variable and setting repopulate options

Now you need to tell the `Main` procedure to use the subclass procedure. Embed this code after opening the window:

```
OrigWndProc = Window{PROP:WndProc}
Window{PROP:WndProc} = ADDRESS(MainSubClassFunc)
```

This code has now assigned `MainSubClassFunc` to handle all external events for the window. There is one last step, which is to trap events sent by the `MainSubClassFunc`. Embed the following code in the `MAIN` procedure's `TakeWindowEvent` embed point, priority 6300:

```
CASE EVENT()
OF EVENT:NIM:MouseRight
  Message('Howdy!')
END
```

Run the application, and right click on the apple in the tray. The message "Howdy" should pop up.

That's the essence of the tray icon. The next few steps will show you how to "minimize the application to the system tray".

Step 4: Minimize the application to the icon tray

Minimizing the application to the icon tray is easy –simply hide the application when the window is minimized and add the icon to the tray; when the application is restored, delete the icon from the tray. Make sure that the `MAIN` window has the `IMM` (immediate) attribute and add the following code in the window's `Iconized` event:

```
0{prop:hide}=1
!Hide the window
Err = Shell_NotifyIcon(NIM_ADD,ADDRESS(NotifyIconData))
!Add an icon to the tray
```

Add this code in the window's Restored event:

```
0{prop:hide}=0
Post(Event:GainFocus)
Err = Shell_NotifyIcon(NIM_DELETE,ADDRESS(NotifyIconData))
```

Change the code in the TakeWindowEvent embed to the following, to pop up a menu over the system tray:

```
CASE EVENT()
OF EVENT:NIM:MouseRight
   Execute Popup('Show me|Say Howdy!|-|Exit ' & |
                 'Application|Cancel')
      0{prop:Iconize}=FALSE
      Stop('Howdy!')
      Post(Event:CloseWindow)
   END
END
```

Run the application, and don't click any buttons. Minimize the app. It should disappear and an apple should appear in your Tray. Right click the apple, and you should have something popping up that looks like Figure 4.

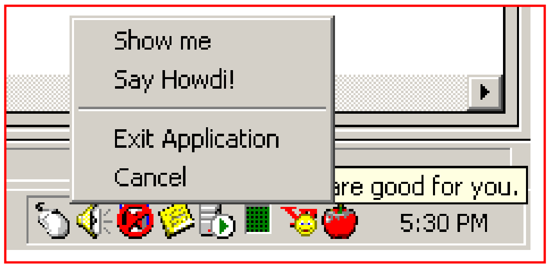


Figure 4. A Clarion popup menu after right-clicking the apple

Try out the various options on the menu. This should illustrate that even though the application is "in the tray" it is still active and you have full control over it.

Sometimes there is a small problem with getting the window to appear at the top of the other applications. Some believe it is a variation in the configuration of the operating system, others say it is an operating system-specific problem. Whatever the reason is, there is always one tool in your toolbox that fixes most things: a hammer. This particular hammer changes the Z-order of a window – it makes sure that the window you apply it to gets the top priority, and Windows complies by reshuffling all the windows to get the specified window to the top.

Add this Windows API prototype to the global map:

```
SetWindowPos(UShort,UShort,Short,Short,Short,Short,|
  UShort),BYTE,PASCAL
```

Your API section should now look like this:

```
MODULE('Whatever')
     MODULE('Whatever')
    LoadImage(Unsigned,*Cstring,Unsigned,Signed,Signed,|
    Unsigned),Unsigned,Pascal,Raw,Name('LoadImageA')

    Shell_NotifyIcon(Unsigned,Signed),Long, Pascal,Name|
    ('Shell_NotifyIconA')

    CallWindowProc(ulong,uLONG,UNSIGNED,UNSIGNED,uLONG),|
    ulong,PASCAL,NAME('CallWindowProcA')

    SetWindowPos(UShort,UShort,Short,Short,Short,Short,|
    UShort),BYTE,PASCAL
END
```

Change the code in the `WindowEvents Restored` embed to this:

```
If SetWindowPos(0{PROP:Handle},-1,0,0,0,0,BOR(2,1)).
```

This blunt instrument is not always necessarily, and does not always work.

However, when you *do* need it and it *does* work, it makes your day that much better!

There are several commercial and freeware templates which add system tray functionality for you, and I would suggest you consider using them. However, sometimes the resulting effect is not exactly what you want, and the knowledge of how the system tray works will enable you to tweak the templates or classes to meet your requirements exactly.

Download the source

James Cooke has been using Clarion since 2.1 days and has been a die hard for "the cause" ever since. He and his family recently moved from South Africa to Texas and is currently working in the banking industry. He spends most of his free time basking in the sun by the pool with a good book or succumbing to that hard-to-kick addiction that persistently haunts the Western cosmopolitan neighborhoods - the yard sale.

Reader Comments

Add a comment

James, On Terminal Server/Citrix we have found the...