Clarion Magazine

Clarion News

- » ClarionTools Spreadsheet Wizard 6 Demo
- » Ingasoftplus Office Schedule
- » StrategyOnline Products at eSellerate
- » CrossTab Wizard 6 Demo Available from ClarionTools
- » iQ-XML 1.17
- » xWhatsNew Class 2.0
- » Simsoft Office Schedule
- » ClarionTools Report Wizard 6 Demo Ready for Download
- » View Wizard 6 Demo Available from ClarionTools
- » Data Mapper 1.22 Released
- » ClarionTools Query Wizard 6 Demo
- » XML Schema Whitepaper
- » Dictionary Assistant 15 Day Trial
- » Data Equity Dictionary Assistant 2.13 Update
- » FullRecord 1.32
- » xQuickFilter 2.18
- » xXPpopup 1.7
- » ClarionTools Presents "The Wizards in Oz"
- » New Versions and Special Offer from Evolution Consulting
- » Dot to End Convertor
- » Huenuelufu Web Site Redesigned
- » Calling PlugIT Customers
- » Free TXD Parser: Clarion Data Mapper
- » Clarion Connection
- » cpTracker Pro Source Sale Extended

- » SealSoft Special Offers
- » d-Document Icons And d-User Icons
- » New Product: J-Fax
- » Huenueleufu's Products 9052 Compatible
- » Evolution Tools April Special
- » Clarion From The Start Source Code
- » Ole Drag and Drop 1.03
- » xPowerManager 1.
- » Icetips Report Previewer 2.2
- » Dutch Clarion Forum
- » Projman Pre-Pre-Alpha
- » Data Ferret 4.01
- » Data Equity Assistants Documentation Update
- » FullRecord Price Increase May 1
- » xToolTip 2.0
- » SetupBuilder 5.4 Build 146
- » CapeSoft World Tour
- » Easy3DStyle 3.00
- » Whitemarsh March Update
- » Keystone Announces OLE Drag and Drop Tool
- » Easy3DStyle 3.00
- » Extended Evaluate C6.3 9051 Compatible
- » xAppWallpaper Manager 2.4
- » xClarionSwitcher 1.06
- » EasyCOMCreator 1.03
- » xXPframe 1.3
- » The Clarion Handy Tools and 9051
- » CPCS for #9051
- » Aussie DevCon Final Announcement
- » Icetips Products and 9051
- » Huenuleufu Products For 9051

- » iQ-XML For 9051
- » RPM, PNet & AFE for 9051
- » Clarion from the Start Special Offer Extended
- » Ingasoftplus Products And C6.3 9051

[More news]

Latest Free Content

[More free articles]

Clarion Sites

Podcast

[Track lists, more podcasts]

Clarion Blogs

Latest Subscriber Content

The Five Minute Developer: Sorting QUEUEs

Clarion's QUEUE structure is useful not just for storing data, but for sophisticated sorting. But by default, QUEUEs are case sensitive. Dave Harms explores several options for case insensitive sorts, including custom sort procedures.

Posted Friday, April 28, 2006

The Five Minute Developer: Displaying QUEUEs

The QUEUE is one of the Clarion language's most powerful, useful, and unique structures. In this installment of the Five Minute Developer, Dave Harms looks at some of the techniques available for displaying QUEUEs in list boxes.

Posted Friday, April 28, 2006

Nifty Window Tricks And Smart DLL Loading

It's an age old question: "How do I use a Windows API function that is not available in all versions of Windows?" The simple answer is that you only call the function if it's available at runtime. You do this by attempting to dynamically load the module where the function resides, and if that succeeds you call the function by address. If you can successfully load the module and get the address of the procedure, it exists in this version of Windows. There's no need to try to identify the version of Windows; either the function exists or it doesn't. In this article Larry Sand demonstrate this technique with some API calls that are only available in Windows 2000 and later, and which allow you to gradually fade in a window and make an area transparent or opaque.

Posted Thursday, April 20, 2006

Aesthetically Pleasing Recursive Updates

Clarion makes it easy to do recursive inserts, where you continue to insert records without going back to the browse each time. It is a matter of but a few mouse clicks. But how about recursive updates? Henry Plotkin looks at the ugly way, and the pretty way.

Posted Monday, April 17, 2006

Mixing Clarion With.NET, Part 6

In the concluding part of this series, Wade Hatler shows how to make COM and C++ wrappers, and demonstrates some nifty ways to spruce up a standard Clarion window via .NET.

Posted Thursday, April 13, 2006

Updating Hot Fields

On a browse window, Hot Fields significantly extend the display of information. Fields for which there is no space in the list control, data from other files (related or not), run time computed data are all nicely handled with Hot Fields. But when Steve Parker experiments with read-only entry fields on a browse, he finds out his users now want to use these fields update values.

Posted Tuesday, April 11, 2006

Multi-User Primary Keys: A Solution

The easiest way of generating primary key values is to use Clarion's autoincrement feature. But when multiple users access the same table/form simultaneously, this can cause problems. Rhys Daniell shows how to take control of primary IDs in MS SQL using a specialized table, two stored procedures, and some Clarion code.
Posted Friday, April 07, 2006

PDF for March 2006

All Clarion Magazine articles for March 2006 in PDF format. Some articles published in March were actually part of the Feb issue, due to that month's schedule disruption.
Posted Tuesday, April 04, 2006

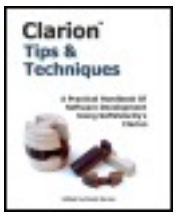
[Last 10 articles] [Last 25 articles] [All content]

Printed Books & E-Books

E-Books

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

Printed Books



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

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

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

From The Publisher

About Clarion Magazine

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

Subscriptions

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

Satisfaction Guaranteed

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

Dave Harms

Clarion Magazine

Clarion News

Search the news archive

ClarionTools Spreadsheet Wizard 6 Demo

The Spreadsheet Wizard 6 demo shows how your users can select a template file with the saved definition and create some pretty compelling output. You can drop Spreadsheet Wizard into your app and dump data off a browse or any process, but there is so much more you can do. ClarionTools has added support for international versions of Excel and any sort of cell formatting. For example, the demos show off some simple formatting to highlight negative numbers in red, this is very simple capability to add. Don't forget that your users can easily change the sort order, select styles, add spreadsheet templates, and lock column headings all from our simple to use Wizard interface. You can also plug in Query Wizard 6, and your users can filter the Spreadsheet output right down to the exact data they are looking for.
Posted Wednesday, April 26, 2006

Ingasoftplus Office Schedule

The Ingasoftplus office will be closed from April 26-28, 2006.
Posted Wednesday, April 26, 2006

StrategyOnline Products at eSellerate

You can now purchase all StrategyOnline products from eSellerate, as well as ClarionShop.
Posted Wednesday, April 26, 2006

CrossTab Wizard 6 Demo Available from ClarionTools

CrossTab Wizard allows your users to find a gold mine in their data, and they can easily do it all by themselves with this product (with a little help from you.) CrossTab Wizard provides a simple way to output a cross-section of your data.
Posted Wednesday, April 26, 2006

iQ-XML 1.17

iQ-XML version 1.17 is now available. Changes include: All 8K field limitations changed to 10K; Changed XML Parser to convert all ASCII coded values (For example: Õ or); Added F10 (Expand/Collapse) in the DebugQueue screen to expand the data element. The 9051 release is compatible with both 9051 and 9052, and does not need a recompile.
Posted Wednesday, April 26, 2006

xWhatsNew Class 2.0

xWhatsNew Class 2.0 is a class and set of extension and code templates to add a "What's New" feature to your application. New in this release: Compatibility with Clarion 6.3 (build 9052); Changes related to frame menu; Compatibility with xToolTip and xToolTipPro.
Posted Wednesday, April 26, 2006

Simsoft Office Schedule

Eric Churton will be away from the office and unable to attend to queries for up to five days.
Posted Wednesday, April 26, 2006

ClarionTools Report Wizard 6 Demo Ready for Download

Report Wizard 6 puts report generation power into the hands of the average user. While it can produce some fairly sophisticated reports, the simple and familiar Wizard interface gets users started quickly and without a lot of support on your part. You can ship report examples (or canned reports) with your app that the user can copy, but cannot modify. The product supports up to nine rows per detail line and your users can add sort orders, create breaks with totals, averages, and counts and so much more!
Posted Wednesday, April 26, 2006

View Wizard 6 Demo Available from ClarionTools

View Wizard 6 adds a number of great new features and fixes many of the problems that plagued older versions. Try out the locator after sorting on any field and see how it uses the current sort field to locate on including the field picture for dates. Check out the new support for styles and icons on the popup menu, don't forget drag and drop column movement.
Posted Wednesday, April 26, 2006

Data Mapper 1.22 Released

You can download the latest build from the web site, or simply use the "Check for Updates" feature from the "Help" menu. This is the first non-beta release. Data Mapper can now be purchased at ClarionShop. You can still use the free version indefinitely, but it is restricted it to two Dictionaries in the Recent Projects list. The licensed version $49 for a 12 month license, which entitles you to unlimited access to all the current and future features, and free updates etc for the next 12 months.
Posted Wednesday, April 26, 2006

ClarionTools Query Wizard 6 Demo

A demo of ClarionTools Query Wizard is now available for download.
Posted Wednesday, April 26, 2006

XML Schema Whitepaper

Steve Stockstill has started a new white paper on using the XSD specification for dictionary diagramming and interfacing Clarion Dictionaries with other development platforms.
Posted Wednesday, April 26, 2006

Dictionary Assistant 15 Day Trial

Data Equity has posted a new 15 day trial/demo edition of Dictionary Assistant in the downloads section.
Posted Wednesday, April 26, 2006

Data Equity Dictionary Assistant 2.13 Update

Data Equity Dictionary Assistant 2.13 is now available. This release includes the

completed integration of a significantly enhanced html help system. There is also a new trial edition in the downloads section. As always you may use the "Check for Updates" option within DA or download the update directly from the web site
Posted Wednesday, April 26, 2006

FullRecord 1.32

FullRecord 1.32 is now available. Changes include: Fields inside dimensioned groups now inherit the dimensions of the group and the group itself is not processed; When storing the record in a memo field, a memo-to-record comparison in clarion 6.2 produced a GPF - fixed. Remember the price increase on May 1st 2006.
Posted Wednesday, April 26, 2006

xQuickFilter 2.18

xQuickFilter 2.18 is now available. New in this version: Compatible with Clarion 6.x (6.3, 6.2, 6.1); Changes in extension template.
Posted Wednesday, April 26, 2006

xXPpopup 1.7

xXPpopup 1.7 is now available. New in this version:; Compatible with Clarion 6.3 (build 9052); Clarion 5 is not supported now (use xXPpopup v1.5); Temporary hiding of menu item; New options; Updated demo.
Posted Wednesday, April 26, 2006

ClarionTools Presents "The Wizards in Oz"

The ClarionTools "Wizards" will be on show at the Aussie "Oz" DevCon 2006 which starts Friday, April 28th. Be the first to see the brand new production of our revamped Version 6 Wizard product line.
Posted Wednesday, April 26, 2006

New Versions and Special Offer from Evolution Consulting

Evolution Consulting has released version 1.10 of Evolution Report Export and Evolution Browse Export, with several improvements and fixes. Special pricing is available until the end of April.
Posted Wednesday, April 26, 2006

Dot to End Convertor

Randy Rogers has a utility available for free download to convert the dot character to END, in both CLW and TXA files.
Posted Wednesday, April 26, 2006

Huenuelufu Web Site Redesigned

The Huenuelufu web site has been completely redesigned for cleaner and wider pages, and a new all-in-one horizontal menu. The site also features a new logo, developed by Leroy Schulz.
Posted Wednesday, April 19, 2006

Calling PlugIT Customers

On 1 Feb this year Gary James at StrategyOnline announced that his company had taken over the development, maintenance and support of PlugIT, from Plugware. If you have ever bought PlugIT (any version, at any time, from anyone), please go to the link above and enter your name and email address. Your information will not be shared with anyone; this is to notify you if you qualify for free updates etc. Also, if you sell Clarion accessories that use PlugIT, and if you are shipping our pwutil.dll with your accessories, please let StrategyOnline know. Gary would like to standardize how this DLL is installed (as per the C3PA), and also ensure that no one installs an older version of it over a newer version.
Posted Wednesday, April 19, 2006

Free TXD Parser: Clarion Data Mapper

Gary James has updated his TXD Parser and renamed it Clarion Data Mapper. Improvements in this build include: Recompiled the APP in Clarion 6.3; Added a splitter bar to the main window; You can now save the diagram to EMF, or WMF; You can now view a Grid, and SnapToGrid; Some other minor improvements.
Posted Wednesday, April 19, 2006

Clarion Connection

The Clarion Connection is temporarily down due to an unsuccessful server transfer.
Posted Wednesday, April 19, 2006

cpTracker Pro Source Sale Extended

The cpTracker Pro Source Code offer for $47, with free cpTracker Pro, has been extended until April 23, 2006. The regular price is $299 (single company use) and $1196 (royalty-free reseller edition).
Posted Wednesday, April 19, 2006

SealSoft Special Offers

You can now purchase the source code for xPowerManager 1.03, SealSoft's free shutdown management utility. xPowerManager uses the "x-Fighers: xXPpopup, xToolTip and the free xFunction library. If you don't yet have xXPpopup and xToolTip, you can get the complete package for $149 (save $58).
Posted Wednesday, April 19, 2006

d-Document Icons And d-User Icons

David Beggs has released two more icon sets. The first is d-Document Icons, based on a document theme. This set costs $US19.99 but (as usual) Clarion developers can use the coupon CPN5676118870 to receive a 50% discount on any of the icon sets. PayPal is also available. The second is a smaller set based on a user theme. This is free to anyone who purchases or has purchased any of the other icon sets. Instructions for download are on the web site.
Posted Wednesday, April 19, 2006

New Product: J-Fax

StrategyOnline has released J-Fax, a template/class wrapper for the Microsoft Fax components. With J-Fax you can send faxes from your Clarion APPs. Price is $39. Requires Windows 2000, XP, or 2003. Clarion 5.5 - 6.3 compatible, ABC & Legacy.
Posted Wednesday, April 19, 2006

Huenueleufu's Products 9052 Compatible

As Huenueleufu's products are all source code, you just have to recompile and you are ready to go. The FinalStep template invalidated a last minute fix to 9052, but SoftVelocity enhanced the RTL to avoid this problem.
Posted Wednesday, April 19, 2006

Evolution Tools April Special

This is a limited time special offer for all the Evolution Consulting products. Prices: Evolution Report Export, $89; Evolution Edit in form, $89; Evolution Excel Import, $49; Evolution Browse Export, $89.

Posted Wednesday, April 19, 2006

Clarion Magazine

The Five Minute Developer: Sorting QUEUEs

by Dave Harms

Published 2006-04-28

While displayed queues are the most familiar to Clarion developers, non-displayed queues have a myriad of uses. For one, queues can be sorted by multiple fields, functioning much like multi-component keys. You sort queues with the SORT function, and you can pass up to 16 queue field names for the sort order (for a grand total of up to 17 parameters, including the queue which is passed first). The default sort is ascending, but you can make any field descending by prefixing a minus character (–) to the field name. You can also put a + character in front for ascending order, and there are no restrictions on the actual order in which the fields appear in the parameter list:

```
SORT(MyQueue,+Field4,-Field2,-Field1,+Field3)
```

Note that reference field types and arrays cannot be used for sorting.

SORT, used alone, only does case sensitive sorts, because it doesn't keep a separate key structure. If you want to do case insensitive sorting, you have several options. One is to create an extra field in your queue with an UPPERed or LOWERed copy of the data you want to sort, and sort on that field.

Another option, as demonstrated in the Clarion Help, is to use a custom sort procedure. I'll reproduce the code here, as there are a few points worth further explanation:

```
PROGRAM
 MAP
  CaseInsensitive(*GROUP A, *GROUP B),SIGNED
 end

Q      QUEUE
Val      STRING(5)
       END

Window WINDOW('Test Sort'),AT(,,116,224),FONT('MS
SansSerif',8,,FONT:regular),IMM,SYSTEM,GRAY,AUTO
       BUTTON('Sort Case Sensitive'),AT(8,3,95,14),USE(?SortCase),LEFT,DEFAULT
       BUTTON('Sort Case INSensitive'),AT(8,20,95,14),USE(?SortNoCase),LEFT
       LIST,AT(6,37,101,179),USE(?List1),FORMAT('7L(2)|M~Val~@s5@'),FROM(Q)

     END

 CODE
```

```
  Q.Val = 'aaaaa'; Add(Q)
  Q.Val = 'AAAAA'; Add(Q)
  Q.Val = 'ddddd'; Add(Q)
  Q.Val = 'DDDDD'; Add(Q)
  Q.Val = 'EEEEE'; Add(Q)
  Q.Val = 'eeeee'; Add(Q)
  Q.Val = 'qqqqq'; Add(Q)
  Q.Val = 'QQQQQ'; Add(Q)
  Q.Val = 'zzzzz'; Add(Q)
  Q.Val = 'ZZZZZ'; Add(Q)
  Q.Val = 'ggggg'; Add(Q)
  Q.Val = 'GGGGG'; Add(Q)

  OPEN(Window)
  ACCEPT
    CASE ACCEPTED()
      OF ?SortCase   ; SORT(Q, Q.Val)
      OF ?SortNoCase ; SORT(Q, CaseInsensitive)
    END
  END

CaseInsensitive PROCEDURE(*GROUP A, *GROUP B)!,SIGNED

  CODE
  IF    UPPER(A) = UPPER(B) THEN RETURN  0
  ELSIF UPPER(A) > UPPER(B) THEN RETURN  1
  ELSE RETURN -1
  END
```

Note that the `CaseInsensitive` procedure is prototyped in the map as taking two groups, passed by address. These parameters are actually queue records, and the example uses the "implicit first field" to do the comparison. That is, you can use the queue name as a synonym for the first field. It is possible to use the sorting procedure to sort on a different field, but you'll need to employ a little WHO and WHAT magic. I'll leave that as the proverbial exercise for the reader (or a later installment).

You can call this function anything you like – just be sure it has the correct prototype, and that it's visible to the `SORT` function. For instance, if you create your `SORT` procedure using the AppGen, you will need to either check the Declare Globally option on the procedure properties, or ensure that the source procedure is in the same module as the procedure that uses it as a parameter to SORT.

As noted in the help, within your custom sort procedure you want to return 0 if the two elements have the same value, 1 if the first element has a higher value than the second, and –1 if the first element is lower than the second. That gives you an ascending order. As you may have guessed, if you reverse the 1 and –1 values you'll get a sort in descending order.

The custom sort procedure is useful when you have data such as numeric strings, Roman numerals, and the like, as noted by Gordon Smith in Custom Queue Sorting. The necessary code has been simplified a bit since Gordon's article, as you can see; among other things, you no longer need the `NAME` attribute on the sort procedure.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995), and has written or co-written several Java books. David produces the Planet Clarion podcast, which he co-hosts with Andrew Guidroz II.

Reader Comments

Add a comment

- » Besides SORT() the other Queue commands (GET, ADD, PUT)...
- » Besides SORT() the other Queue commands (GET, ADD, PUT)...
- » Cool, thanks Carl! Dave

Clarion Magazine

The Five Minute Developer: Displaying QUEUEs

by Dave Harms

Published 2006-04-28

The QUEUE is one of the Clarion language's most powerful, useful, and unique structures. You may take the QUEUE for granted, but if you try to duplicate QUEUE functionality in another programming language, you'll quickly come to appreciate what a wonderful construct is the QUEUE.

In this installment I'm not going to focus on the mechanics of reading and writing queues – you can get all that from the Clarion help. Instead, I'll outline a few key points about displaying queues.

The FROM attribute

The most common use of a queue is to display data in a list box. To bind a QUEUE to any list box, you simply specify the QUEUE's label as the LIST's FROM attribute (and note that you can also supply a string constant or a GROUP as the FROM attribute). Here's an example of a list box displaying a queue called Queue:Browse:1 (this is a highly abridged declaration, as I'll explain in a moment):

```
LIST,AT(8,20,342,124), FROM(Queue:Browse:1)
```

FROM binds the queue to the list box, so that all of the data in the queue is automatically displayed. If you don't specify any formatting information, then the list box will simply display all of the queue's fields, in order, as best as it can. In most cases, however, you'll want to specify the formatting, via the FORMAT attribute, or the list box formatter in the

IDE (which creates a format string). Here's a complete list box declaration with the FORMAT attribute:

```
LIST,AT(8,20,342,124),USE(?Browse:1),IMM,HVSCROLL,|
  MSG('Browsing Records'),|
  FORMAT('64R(2)|M~AdID~C(0)@n-14@80L(2)|' &|
  & 'M~Description~@s255@64R(2)|M~NameID~C(0)' &|
  & '@n-14@44D'(16)|M~Amount~C(0)@n10.2@20L(2)' &|
  & '|M~Paid~@s1@'),FROM(Queue:Browse:1)
```

This is an example of a fairly small format string, as the list box has only five fields and no icons, colors, or other special modifiers. And there really are a lot of modifiers – just check out FORMAT in the Help.

PROP:FORMAT

You really don't want to have to write format strings by hand, which is why there's a list box formatter. On the other hand, there are sometimes good reasons to fool around with format strings. You might want to create a set of commonly used format strings, and let your user choose the one appropriate to the task (with different field ordering, colors, icons, etc.). It's easy enough to do - you can read and write format strings with PROP:FORMAT. This was, in fact, one of the very first features that David Bayliss demonstrated to Clarion developers when Clarion for Windows made its debut at the Boca Raton DevCon.

Try this: Find any list or browse box in your application, and double-click on the control in the editor to bring up the embed list. In the All Events embed, place the following code:

```
0{prop:text} = ?listBoxEquate{PROP:FORMAT}
```

That line of code will cause the format string to be displayed in the current window's title bar. Try dragging the list's columns around and watch the format string change.

PROPLIST

Working directly with a format string can be quite complicated; it's a lot easier to use the PROPLIST properties (see FORMAT, again) to read and write the properties of any one individual column. For instance, this code will set the title of the second column:

```
?listBoxEquate{PROPLIST:Header,2} = 'Column 2'
```

In most cases PROPLIST properties are read/write. For instance, to read the current header you use this code:

```
MyString = ?listBoxEquate{PROPLIST:Header,2}
```

PROPLIST properties let you read and/or control colors, styles, decimal settings, tips, positioning, grouping, headers, icons, mouse clicks, borders, and tree appearance.

There are a number of products out there that deliver browse customization via PROPLIST and FORMAT. I've always been a fan of Xplore, which was originally developed by Brian Staff, and which is now sold and supported by IceTips.

Incidentally, any translation utility that handles browse headings will also work via the format string.

Spend some time reading the docs for FORMAT – you'll be amazed at what the format string can do for your applications.

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of *Developing Clarion for Windows Applications*, published by SAMS (1995), and has written or co-written several Java books. David produces the Planet Clarion podcast, which he co-hosts with Andrew Guidroz II.

Reader Comments

Add a comment

Clarion Magazine

Nifty Window Tricks And Smart DLL Loading

by Larry Sand

Published 2006-04-20

It's an age old question: "How do I use a Windows API function that is not available in all versions of Windows?" The simple answer is that you only call the function if it's available at runtime. You do this by attempting to dynamically load the module where the function resides, and if that succeeds you call the function by address. If you can successfully load the module and get the address of the procedure, it exists in this version of Windows. There's no need to try to identify the version of Windows; either the function exists or it doesn't.

In this article I'll demonstrate this technique with some API calls that are only available in Windows 2000 and later, and which allow you to gradually fade in a window and make an area transparent or opaque.

Getting started

In a previous Clarion Magazine article I described how to prototype procedures with a little compiler trick that allows you to call them by address. That article describes a class called `LoadLibClass`; I'll make use of that class here. So the first thing you'll need is to review that article and download the associated source zip.

> **NOTE: Changes required to LoadLibClass**
>
> The `LoadLibClass` header from the original article used `_ABCLinkMode_` and `_ABCDLLMode_` to control linking and to specify whether or not the class was exported from a DLL. If you plan to use this code in Legacy, or if your ABC classes are exported from a DLL, you will need to make a change to the class definition.

```
LoadLibClass   CLASS,TYPE,MODULE('LoadLib.clw'),|
                          LINK('LoadLib.clw',_ABCLinkMode_),|
                          DLL(_ABCDllMode_)
And change them to:
LoadLibClass   CLASS,TYPE,MODULE('LoadLib.clw'),|
                          LINK('LoadLib.clw',1),|
```

```
DLL(0)
```

This change means that the class is always linked into the module where the header file is included. You may also change the Link and DLL attributes to use equate constants. In Clarion 6 you must define these in the "Project Defines". If you attempt to use equates in your code when using Clarion 6 to define these values, your program will GPF when the constructor fires.

Layered windows

One use of layered windows is to make a window fade into view. Your program can accomplish this via the native alpha blending capabilities of Windows 2000 and higher. Alpha blending is a technique that allows you to combine two images and specify the transparency of either image.

Windows 2000 introduced transparent layered windows. A window created in Windows 2000 and higher has an extended style bit identified by the constant `WS_EX_LAYERED`. Note that this style does not work with MDI child windows. When this bit is set, you may modify the transparency through the Windows API `SetLayeredWindowAttributes` API call. Because `SetLayeredWindowAttributes` is only available on Windows 2000, Windows XP, and Windows Server 2003 you will need to load it at runtime.

Before you can use `SetLayeredWindowAttributes`, you must have a way to set the extended style bit for your window. To do this you use the Windows API functions `GetWindowLong` and `SetWindowLong`. They're defined like this on MSDN:

```
LONG GetWindowLong(
HWND hWnd,
int nIndex
);
LONG SetWindowLong(
HWND hWnd,
int nIndex,
LONG dwNewLong
);
```

These translate to the following Clarion prototypes:

```
GetWindowLong(UNSIGNED hWnd,|
              SIGNED nIndex |
              ),Long, Pascal, Name('GetWindowLongA')
SetWindowLong(UNSIGNED hWnd, |
              SIGNED nIndex, |
              Long dwNewLong |
              ),Long, Pascal, Proc, Name('SetWindowLongA')
```

`GetWindowLong` takes two arguments. The first is the `hwnd` or handle to the window. All of the Windows API functions discussed in this article use this value to identify the window that they wish to

change. Clarion has two handle properties for a window, the handle to the client area (`Window{Prop:ClientHandle}`) and the handle to the entire window including the non-client area (`Window{Prop:Handle}`). This example uses the hwnd returned by `Window{Prop:Handle}`.

The second parameter, `nIndex,` is a constant that describes what information you wish to get from the window. In this case, it's the extended style data identified by the constant `GWL_EXSTYLE` (decimal value -20). You can find the value of these constants in the Windows header files from Microsoft. `GetWindowLong` returns the requested long value, or zero if there was an error.

`SetWindowLong` takes three arguments. The first two are identical to `GetWindowLong`. The third parameter `dwNewLong` is the information that you want to store in the `WindowLong` identified by the `nIndex` parameter. The return value is the previous value or zero if the function failed. You should always use `GetWindowLong` to first get the current data and then make your changes to that value and put it back with `SetWindowLong`.

Both Clarion prototypes of `GetWindowLong` and `SetWindowLong` require the name attribute with an "A" appended to their name to signify that you'll use the ANSI versions. There are UNICODE versions of these functions that have a "W" appended to their names.

`GetWindowLong` and `SetWindowLong` take care of the requirement to change the value of the extended style bit that allows Windows to draw the layered windows. To change the window's transparency or translucency levels you must call the `SetLayeredWindowAttributes` function.

`SetLayeredWindowAttributes` is defined on MSDN as:

```
BOOL SetLayeredWindowAttributes(
HWND hwnd,
COLORREF crKey,
BYTE bAlpha,
DWORD dwFlags
);
```

Here's the Clarion prototype for this function:

```
SetLayeredWindowAttributes( UNSIGNED hwnd,  |
                UNSIGNED crKey, |
                BYTE bAlpha,    |
                UNSIGNED dwFlags|
        ),BOOL, Pascal, Proc, DLL(_fp_)
```

`SetLayeredWindowAttributes` takes four arguments and returns a nonzero value for success. The first parameter is `hwnd` or handle to the window. The next two parameters `crKey` and `bAlpha` are mutually exclusively. `crKey` is a four byte value that contains a Clarion color equate or an RGB value in the same format as the Clarion color equates. `bAlpha` is a byte value, 0 to 255 that represents 0 to 100% opacity. And the fourth parameter `dwFlags` accepts a constant that tells the function if it should use the `crKey` or `bAlpha` parameter.

It's important to note that using `bAlpha` makes the entire window transparent to the specified degree, and `crKey` only makes the specific color completely transparent. There's no alpha blending choice when using the `crKey` parameter. Note that whenever the window *or* color is completely transparent, all mouse events are translated to the window showing through that area. More about that later.

`SetLayeredWindowAttributes` returns nonzero when it succeeds and zero for failure. Notice that this prototype has the `DLL(_fp_)` attribute added to the prototype. This is necessary for its use with the `LoadLibClass` from the previous article.

In your global data section you'll need the following four constants

```
LWA_COLORKEY  Equate(00000001h) !use the crKey parameter
LWA_ALPHA     Equate(00000002h) !use the bAlpha parameter
GWL_EXSTYLE   Equate(-20)       !GetWindowLong Extended Style
WS_EX_LAYERED Equate(00080000h) !Window Style Extended Layered
```

Include the header file for the `LoadLibClass` like this:

```
 Include('LoadLib.inc'),Once
```

The code in your global map for the three Windows API functions should look like this:

```
  Map
    Module('Win32API')
      SetLayeredWindowAttributes(UNSIGNED hwnd, UNSIGNED crKey, |
             BYTE bAlpha, UNSIGNED dwFlags),BOOL, Pascal, Proc, DLL(_fp_)
      SetWindowLong(UNSIGNED hWnd, SIGNED nIndex, Long dwNewLong),|
             Long, Pascal, Proc, Name('SetWindowLongA')
      GetWindowLong(UNSIGNED hWnd, SIGNED nIndex),Long, Pascal, |
             Name('GetWindowLongA')
    End
  End
```

Don't forget to declare your function pointer variable for `SetLayeredWindowAttributes`:

```
fpSetLayeredWindowAttributes Long, |
  Name('SetLayeredWindowAttributes')
```

If you're confused about this function pointer variable declaration, please re-read the previous article.

Now that you have the necessary Windows API functions prototyped you'll need an instance of the `LoadLibClass`. I've labeled the object User32, since the `SetLayeredWindowAttributes` function is located in the User32.dll library on Windows 2000 and greater.

```
User32   LoadLibClass
```

As soon as the program loads, attempt to load the library and assign the address of

SetLayeredWindowAttributes to the function pointer variable:

```
fpSetLayeredWindowAttributes = 0
If llcUser32.LlcLoadLibrary('User32.dll', |
  Method:GetModuleHandle) = 0
  fpSetLayeredWindowAttributes = |
    llcUser32.LlcGetProcAddress('SetLayeredWindowAttributes')
End
```

After this code executes, your function pointer equals zero if SetLayeredWindowAttributes is not available in the version of Windows executing the code. Whenever you want to call SetLayeredWindowAttributes you only need to test if the function pointer is *not* zero. If you do call SetLayeredWindowAttributes when the value of the function pointer is zero, your program will GPF. Don't do that.

Once you have this setup work out of the way, you can call GetWindowLong, SetWindowLong, and SetLayeredWindowAttributes to make the window fade into view as it opens. This code can be in any window procedure in your program that has the User32 object and fpSetLayeredWindowAttributes function pointer variable in scope.

In your procedure declare a Byte for the alpha blend value, a Long for the ExtendedStyle information, and an UNSIGNED for the hwnd. You can use the return value from Prop:Handle in the function calls without assigning it to an integer variable. However, Clarion returns all properties as strings and the code passes the handle many times. This code allows Clarion to auto-convert the string to an integer once in the assignment.

```
windowAlphaBlend  Byte,Auto
exStyleLong       Long,Auto
thisHWnd              UNSIGNED,Auto
```

As soon as the window is opened and before any other code executes you can turn on the layered extended style bit and change the alpha blend value via the SetLayeredWindowAttributes function. This code makes the window fade into view:

```
Open(W)
thisHWnd = W{Prop:Handle}
If fpSetLayeredWindowAttributes <> 0
  exStyleLong = GetWindowLong(thisHWnd, GWL_EXSTYLE)
  exStyleLong = BOR(exStyleLong, WS_EX_LAYERED)
  SetWindowLong(thisHWnd, GWL_EXSTYLE, exStyleLong)
  Loop windowAlphaBlend = 0 to 255 by 5
    Yield
    SetLayeredWindowAttributes(thisHWnd, 0, |
          windowAlphaBlend, LWA_ALPHA)
    Display
  End
  SetWindowLong(thisHWnd, GWL_EXSTYLE, |
    BXOR(exStyleLong, WS_EX_LAYERED) )
```

```
    Display
  End
```

First notice that the calls to GetWindowLong, SetWindowLong, and SetLayeredWindowAttributes are all wrapped in the if statement; If fpSetLayeredWindowAttributes <> 0. None of this code is necessary if the version of Windows on the computer doesn't support SetLayeredWindowAttributes. When this code executes on a computer with Windows 98, the window will open normally and computers with Windows 2000 or better, the window fades into view.

To accomplish this, the code calls GetWindowLong with the handle to the window specifying the GWL_EXSTYLE index constant to retrieve the window's extended style long integer. Next the WS_EX_LAYERED window style bit is turned on by ORing the value returned by GetWindowLong (for more on combining values using bitwise operations, see the Clarion Help for BAND, BOR, and BXOR). The ExStyleLong is sent back to the window by calling SetWindowLong with the same first two parameters as GetWindowLong and the ExStyleLong.

Now that the extended window style is set to allow layered windows, you can call SetLayeredWindowAttributes. Remember that the bAlpha argument of the SetLayeredWindowAttributes expects a value from 0 to 255 representing 0 to 100% opacity.

Now it's time to fade in the window. The loop calls Yield, SetLayeredWindowAttributes, and Display. Yield allows other programs some processor time, and SetLayeredWindowAttributes is called with the handle to this window, 0 for the crKey (this isn't used), the windowAlphaBlend value, and the constant LWA_ALPHA. The LWA_ALPA constant tells SetLayeredWindowAttributes to ignore the crKey parameter and use the bAlpha parameter instead. Then the Display statement causes the window to repaint without entering the Accept loop. Without the Display statement, you'll never see the effect of the alpha blending.

Finally, when the loop completes its execution, SetWindowLong is called again to turn off the WS_EX_LAYERED bit of the extended style long by exclusive ORing it with the style bit. Using BXOR in this manner will toggle the bit on and off. Normally you'd need to test that the bit was on with BAND before executing the BXOR so you didn't accidentally turn it back on. This code isn't testing for this condition because the WS_EX_LAYERED bit was just set on by the earlier call to SetWindowLong. Turning this style bit off is necessary because Windows allocates a significant amount of memory for the alpha blending and redirection. Turning this bit off frees those resources.

If you wish to leave your window displayed transparently, do not turn the WS_EX_LAYERED extended style bit off.

Compile and run the code included in the download for this article (SetLayAb.prj and SetLayAb.clw). On Windows 2000 and higher you should see the window fade into view. On Windows 95, 98, and ME you should see the window opened normally. Go ahead and change the step or limit for the windowAlphaBlend values. Comment out the final call to SetWindowLong so the Extended style bit is not turned off to see what happens with final windowAlphaBlend values less than 255.

"Look, there's a hole in my window"

Now that you know how to use alpha blended layered windows it's time to look at how to make a single color transparent. Remember that an alpha blended layered window affects the transparency of the entire window. In contrast, use of the color key transparency makes one color completely transparent. This transparency is the same as if you use an alpha blend value of zero. The difference is that it only affects one color. If you did specify an alpha blend value of zero, your window would disappear. It has the same effect as hiding the window.

`SetLayeredWindowAttributes` allows you to make a single color completely transparent by passing an RGB color as a `COLORREF` in the `crKey` parameter. It is no coincidence that the byte order in the Clarion color equates are identical to a `COLORREF`. If you read the bytes of a `COLORREF` from left to right they are `Reserved`, `Blue`, `Green`, and `Red`. This is different from the Windows API data type `RGBQUAD` that has the bytes in this order: `Reserved`, `Red`, `Green`, and `Blue`. Because a `COLORREF` and Clarion color equate are identical, you may use a Clarion color equate (like `COLOR:Blue`) to define the transparent color. For a color that isn't described by one of Clarion's color equates, declare an `UNSIGNED` or a `LONG` variable with a group over it like this:

```
myBGRcolor      Long
BGRGroup        Group,Over(myBGRcolor)
red           Byte
green         Byte
blue          Byte
reserved      Byte
            End
```

The group declared over the `UNSIGNED` allows straightforward access to the bytes that represent red, green and blue. Each RGB component has a range of 0 to 255. Zero means none of that color and 255 means 100% of that color. For example, to use the color green make these assignments:

```
BGRGroup.red = 0
BGRGroup.green = 255  !or 0FFh
BGRGroup.blue = 0
```

To use this, pass `myBGRcolor` for the `crKey` parameter of `SetLayeredWindow` attributes.

Time to put this together and drill some holes in your window. Consider this code used to turn on transparency for a color key:

```
If fpSetLayeredWindowAttributes <> 0
  exStyleLong  = GetWindowLong(thisHWnd, GWL_EXSTYLE)
  SetWindowLong(thisHWnd, GWL_EXSTYLE, |
    BOR(exStyleLong, WS_EX_LAYERED))
  SetLayeredWindowAttributes(thisHWnd, COLOR:Red, 0, |
    LWA_COLORKEY)
End
```

You should be familiar with the first three lines of code already. First and foremost this code tests that the function pointer variable is not equal to zero before proceeding. Next it gets the extended style long and turns on the WS_EX_LAYERED style bit. Finally, it calls the SetLayeredWindowAttributes function. Instead of passing a value for bAlpha, the call passes COLOR:Red for the crKey parameter and zero for the bAlpha parameter; dwFlags is set to the constant LWA_COLORKEY. This constant tells SetLayeredWindowAttributes to use the COLORREF in crKey instead of the bAlpha value. This causes Windows to draw the layered window with anything colored red as completely transparent.

The really interesting thing is that not only is the area transparent, but it's as if Windows drilled a hole in your window wherever the color red used to be painted. If you move your mouse pointer over this area the cursor will change depending on the window below your transparent window. All mouse actions in the transparent area affect the window displayed in that transparent area.

Now to turn the transparency off you'll use this code:

```
If fpSetLayeredWindowAttributes <> 0
  exStyleLong  = GetWindowLong(thisHWnd, GWL_EXSTYLE)
  If BAND(exStyleLong, WS_EX_LAYERED) <> 0
    SetWindowLong(thisHWnd, GWL_EXSTYLE, |
          BXOR(exStyleLong, WS_EX_LAYERED) )
  End
End
```

The only thing different here is that the code tests if the WS_EX_LAYERED extended style bit is turned on before toggling it off. This is done using binary AND with the extended style long and the WS_EX_LAYERED style bit. When the return value of this BAND statement is non zero, the bit is already set and it's safe to exclusive OR (BXOR) the extended style long. If you didn't test this and the bit was off, you'd end up turning it on. BXORing a bit in this manner toggles it on and off.

Try it out, load the project (SetLayTn.prj and SetLayTn.clw) and compile it. Then press the Make box Transparent button. That very red box is replaced with the window underneath. Move your mouse over the transparent area and click.

Wrapping it up

Both examples presented in this article are projects, not apps. I prefer to unit test small bits of code in a simple hand coded project because it allows me to see all of the code in one block. However, many Clarion programmers never look at the source code of their applications except in the embed window or embeditor of the IDE. With that in mind, I've included a Clarion 6 Legacy example that shows a template-generated splash window that fades away as it closes.

The next time you're faced with deciding whether or not to include a feature that's only available in a newer version of Windows, you don't have to omit it to maintain compatibility. Simply use this method to conditionally implement it in your program so you'll have the best of both worlds. You won't be able to use this strategy for every new feature, but it will add another tool to your toolbox.

Download the source

Larry Sand is an independent software developer who began programming with Clarion in 1987. In addition to normal database development, he specializes in connecting Clarion to external devices like SCUBA diving computers, kilns, and satellite transceivers used in medical helicopters. In other lives, he sailed Lake Superior as the owner/operator of shipwreck SCUBA diving tours and later as a Master for the Vista Fleet. When Larry is not programming you'll find him messing about in boats, or with boats.

Reader Comments

Add a comment

Clarion Magazine

Aesthetically Pleasing Recursive Updates

by Henry Plotkin

Published 2006-04-17

Clarion makes it easy to do recursive inserts, where you continue to insert records without going back to the browse each time. It is a matter of but a few mouse clicks.

In either template set, from a Form’s Procedure Properties window, press "Messages and Titles." On the next window, click the drop down next to "After successful insert" to select "Insert another record." See Figure 1.

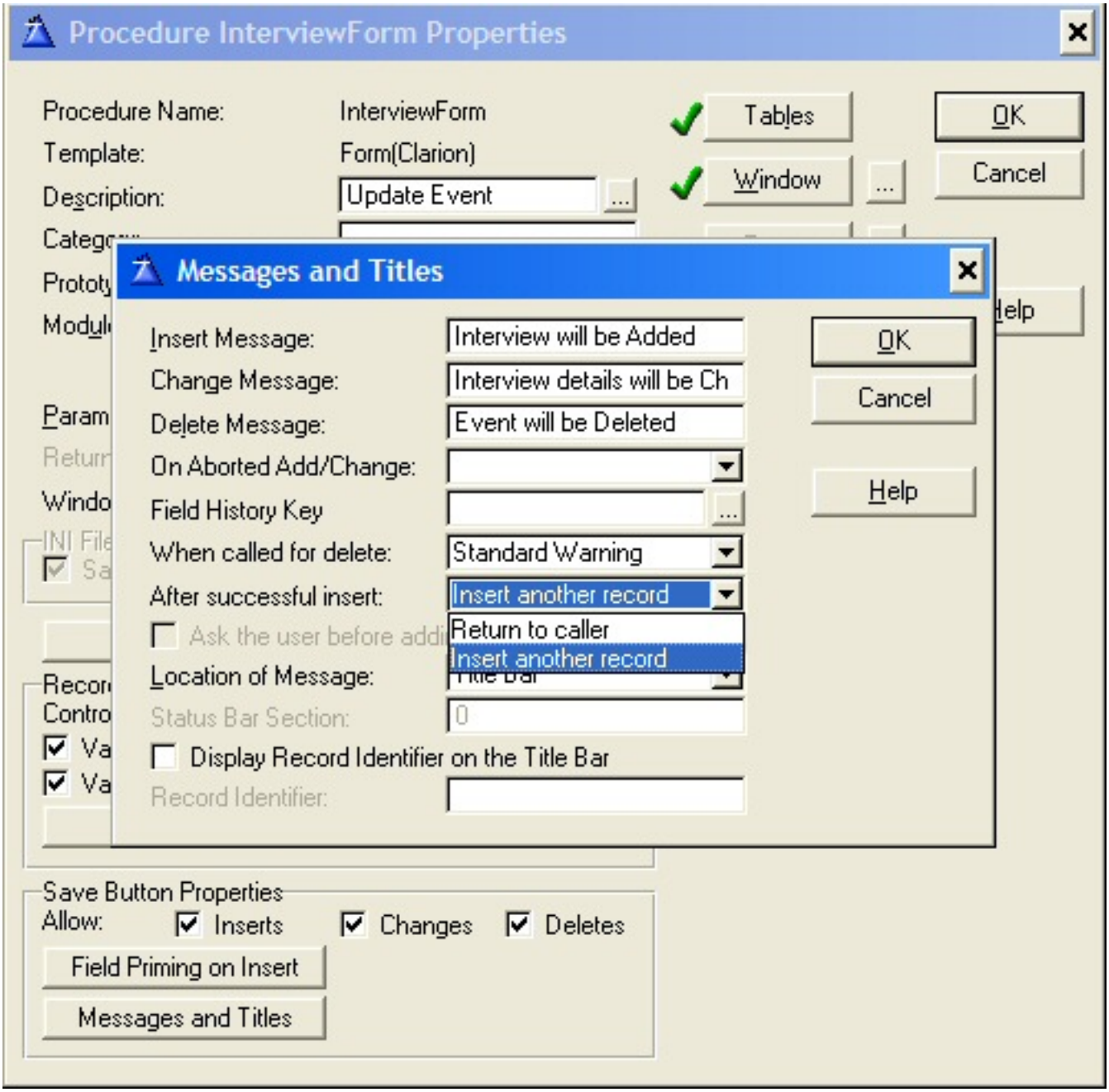


Figure 1. Setting up template for recursive inserts

But I need to do recursive updates. That is, I need to do a number of updates without returning to a browse. In fact, I need to access the form without calling a browse at all: the record to be updated is selected on the form itself via lookup fields. See Figure 2.

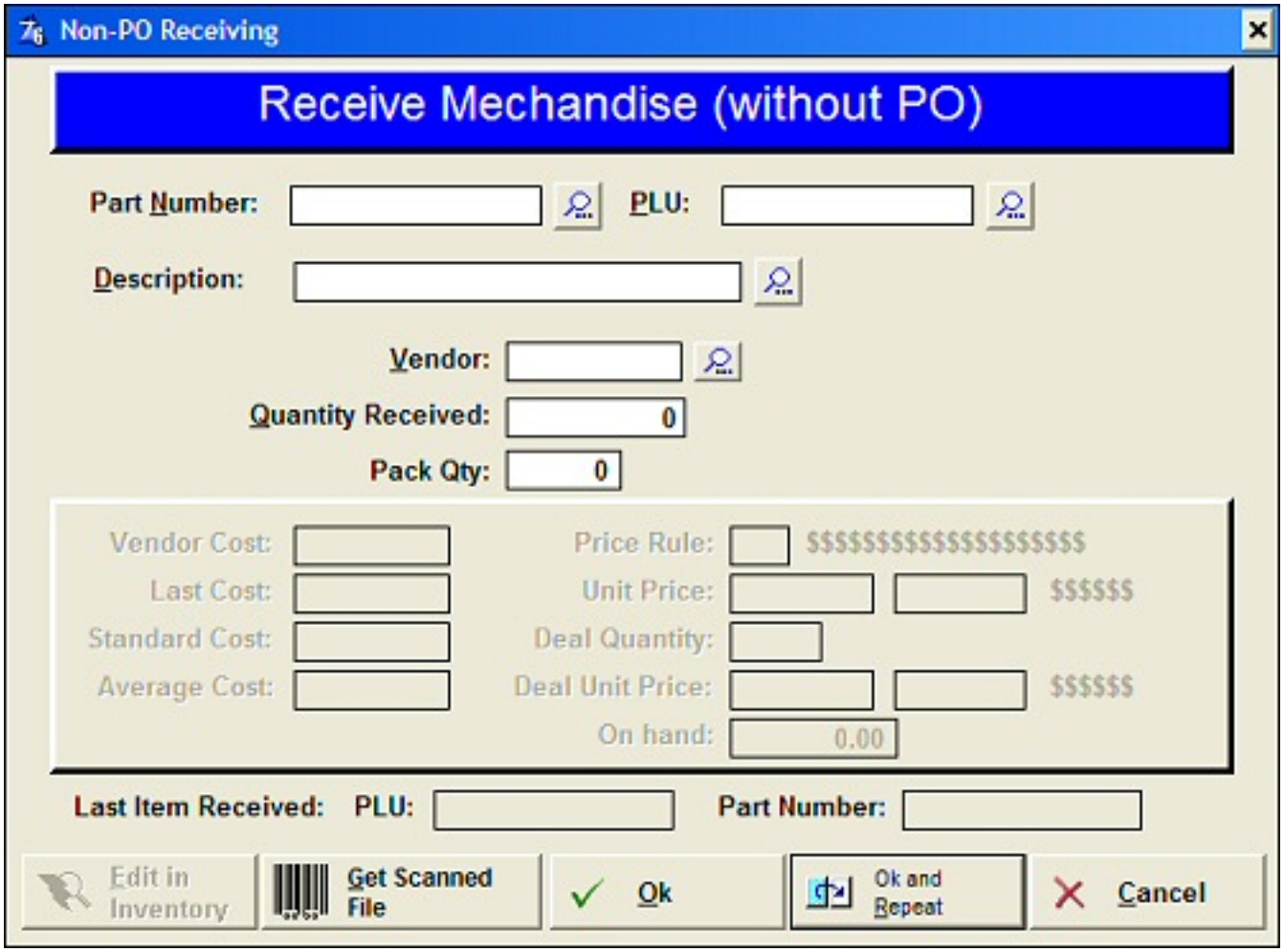


Figure 2. Form used in recursive updates (view full size image)

This form is used to receive merchandise purchased without a purchase order. Each of the first three fields, Part Number, PLU and Description, are lookup fields. From any of these fields, the user may enter or look up the inventory item.

The inventory file is the primary file for this form. It is designed this way because several inventory fields are updated when inventory is received. Data updated include: quantity on hand, year to date quantity received and date last received. A receiving history file is also updated but this is an add. The form procedure is designed this way so that Inventory I/O is automatic; a window procedure, perhaps, would have been better.

There is no direct provision for recursive changes in either the Clarion or ABC templates.

Typically, I call a form without a browse via a Source procedure. To do so recursively, I use a loop and set `GlobalRequest` immediately before the form is called:

```
Loop
```

```
      Clear(INV:Record)
      Clear(REC:Record)
      GlobalRequest = ChangeRecord
      If ReceiveNonPO() = 0
        Break
      End
    End
```

Because `ReceiveNonPO` updates an existing `Inventory` record, the `Inventory` buffer must be cleared. Because receiving history is also updated, to be safe, its buffer is cleared. `GlobalRequest` is set to `ChangeRecord`. The update form is called. It returns a value and so long as it does, the loop iterates.

Because this form is specifically intended to update records without a browse, there are a few other unusual housekeeping items to attend to.

First, I intend to update `Inventory` but I also need to look up an `Inventory` record. Therefore, as you will see in the dictionary for the demo app, I create an `Alias` of `Inventory`. The `Alias` is used for the lookup (see the `BrowseInventory` procedure in the demo app). This is probably overkill, since I really don't care about the `Inventory` buffer at the time the user calls the look up. But, generally speaking, it is safer practice.

Second, when the look up is completed, the aliased file's buffer is current but the primary file buffer is not. Therefore, I get the record:

```
INV:PLU = INV1:PLU
Access:Inventory.Fetch(INV:PLUKey)
```

This technique works quite well. The inventory fields are updated and receiving history is too (see TakeCompleted, Before Parent call).

The problem is that whenever the form is completed using the "Ok and Repeat" button, any window previously opened shows for a moment. Since this procedure is called from a "window menu" (see Figure 3, below), at least one window will "flash" each time the loop cycles.

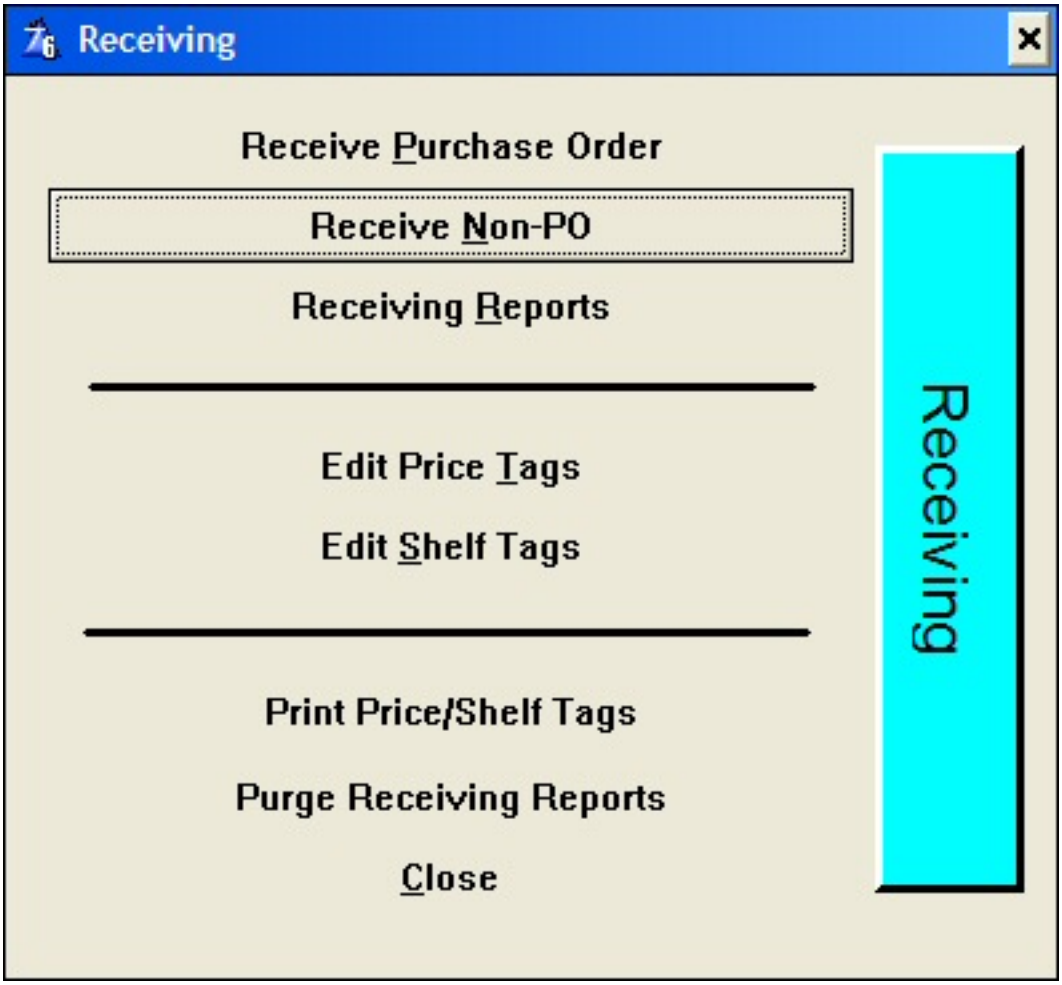


Figure 3. The window that "flashes"

To see this in action, open the demo app (it is written in 6.2). Select Receiving from the top menu. Select Traditional Loop from the window used as a menu.

This is a problem because it isn't pretty. Some users comment on it. I suppose it looks unpolished, unprofessional. And, recursive inserts don't "flash," so why should recursive updates?

So, what does recursively inserting know that my loop does not?

Recursive Inserts

When I select "Insert another record" for "After successful insert," as shown in Figure 1, the Form template changes

```
SELF.InsertAction = Insert:None
```

to

```
SELF.InsertAction = Insert:Batch
```

This assignment is in `ThisWindow.Init` just before the window is opened. There are no further references to either `InsertAction` or `Insert:Batch` in the generated code.

`Insert:Batch`, however, is a property of the `WindowManager` class, found in abwindow.clw. It is used in the `InsertAction` procedure. If `InsertAction` is `Insert:Batch`, the buffers are cleared, field priming is performed and the first field is selected.

Nothing surprising there. Neither is there anything particularly helpful.

However, searching further in abwindow.clw, there is another reference to `Insert:Batch` Oddly enough, it is in the `WindowManager.ChangeAction` method. Change action? Well, if there is an autoincrement key, `Self.OriginalRequest` *might* indeed be insert record.

But this contributes nothing pertinent to the task at hand so far as I can see.

Looking further down this segment, I see a property, `SELF.BatchProcessing` that I do not find in C5.5. Searching abwindow.clw for this reveals some interesting code in the `PostCompleted` method:

```
IF SELF.OriginalRequest = ChangeRecord OR +
   SELF.OriginalRequest = InsertRecord OR +
   SELF.OriginalRequest = ViewRecord THEN
   IF SELF.OriginalRequest = ViewRecord AND
     NOT SELF.BatchProcessing THEN
      POST(EVENT:Completed)
   END
   SELECT()
ELSE
  IF NOT SELF.BatchProcessing THEN
     POST(EVENT:Completed)
  ELSE
     SELECT()
  END
END
```

which appears to cycle back (`SELECT()`) if `SELF.BatchProcessing`. Searching for further occurrences of `SELF.BatchProcessing` reveals:

```
IF SELF.OriginalRequest = ChangeRecord THEN
  IF NOT SELF.BatchProcessing THEN
    CASE SELF.ChangeAction
    OF Change:Caller
      POST(Event:CloseWindow)
    OF Change:Batch
    END
```

While this code does not appear to do anything at this time, it clearly implies that `SELF.BatchProcessing` has an effect (probably in the first of these two snippets).

SELF.BatchProcessing

In the demo app, I added

```
SELF.BatchProcess = True
```

to the Accepted embed of the "Ok and Repeat" button.

In the demo, select "Test 1" from the menu. After selecting an inventory item, entering a received quantity and clicking "Ok and Repeat," nothing appears to happen. If I then select another inventory record and enter a quantity, again nothing.

However, when I click "Cancel" and examine the Inventory file in Topscan, each record *has* been updated correctly. Receiving History has also been updated correctly.

Okay, I understand this. The buffers and display have not been cleared.

I know that file I/O happens in `PARENT.TakeCompleted()`. So, after that, it is safe to:

```
Clear(INV:Record)
Clear(REC:Record)
ThisWindow.Reset
Clear(LOC:Vendor)
```

```
Select(SELF.FirstField)
```

This clears the buffers and the local vendor field. The reset ensures that the form's fields are blanked. Finally, the first entry field is selected.

In the demo app, select "Test 2" from the menu to see this in action. It is just what is wanted: a flicker free batch update.

Summary

The `BatchProcessing` property is not available in 5.5. It was introduced in 6.x, somewhere. Therefore, this technique is not applicable to pre-6.x apps. Unfortunate.

Still, it is uncanny what a few hours of wandering about the `WindowManager` will do....

Download the source

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

Reader Comments

Add a comment

Clarion Magazine

Mixing Clarion With.NET, Part 6

by Wade Hatler

Published 2006-04-13

In Part 2 of this series I showed how to use the OLE control to communicate between Clarion and .NET. This control is a bit untidy for a couple of reasons:

1. It's slow because it uses late binding *and* it has to interpret a string for every pass, plus look up the dispatch address at runtime.
2. The syntax is pretty clunky because it has to be in a string. These are a bit of a hassle to generate, but worse yet they're brittle. Change an interface and the compiler doesn't help you at all with finding code that's broken, and you have to escape the string if it has any double quotes in it.
3. You have to have a window to use it, although this isn't that big of a deal most of the time as you can create a control in whatever window happens to exist, and destroy it when you're done.
4. You can't pass or return arrays, groups or other complex structures.

Having said all that, it isn't as bad as you might think. The speed isn't actually bad at all, and for most application I don't think it's an issue. On my 1.8 MHz machine, I passed and returned a 25 byte string 10,000 times. This took 360 ms, or .036 ms/call. A native Clarion procedure that does the same thing takes 30 ms, so it's definitely slower but not as much as I expected. It's pretty rare where burning 300 ms for 10,000 calls is going to be a real performance problem. The syntax can be greatly improved by writing a small wrapper function to generate the actual OLE string, which isn't really rocket science. Earlier versions of Clarion had severe size restrictions on how big of a string you could pass, but that seem to be gone. I've passed and returned 10MB strings. All in all, the OLE control isn't that bad, and it's much better than I expected, considering how badly it behaved in previous versions or Clarion.

If you want to improve things though, there are a couple of options.

1. Clarion has support for native COM access... sort of. What it has is really clunky and ugly compared to a language like VB, but it does work and it works pretty well once you have the proper prototypes. You obviously wouldn't use the OLE control for something like ADO access.
2. You can create wrappers in Managed C++ that expose your .NET methods as ordinary procedures. This is about as fast as calling a native Clarion procedure, but it's a bit more

trouble to write. This method also allows you to call .NET from languages that support API calls but don't support COM.

Making a COM wrapper

There are a couple of ways to make a COM wrapper for Clarion. You can do it using brute-force hand code if you're feeling up to it, (which I rarely am). There are numerous articles on this subject in this magazine. You can search here. A better alternative is to spend 189 bucks for the Ingasoft EasyCom2INC, which will do the heavy lifting for you. I haven't actually used it all that much, but it seems to work pretty well if you set things up right. The attached Interop05 example is the really simple Interop01 from the first segment in disguise, plus a bunch of speed tests.

EasyCom2Inc builds its output from an IDL file. IDL is a Microsoft Standard text language used to define interfaces. You can generate IDL files from any COM DLL. To get the IDL file requires a two step process.

1. First, generate a TLB (Type Library) file from your assembly. Generate the TLB using `REGASM.exe`, which I covered in an earlier segment. Run it with the name of your DLL and the /t switch (e.g. `regasm /t Interop05.dll`). You only need to run it when you make a change that breaks the interface, such as adding or removing a method, or changing method signatures. When it's done, you will have `Interop05.tlb`.
2. Next, open the generated TLB file using `OleView` (it is part of Visual Studio going back to VB 4 at least), and give it the name of the generated TLB file (e.g. `OleView Interop04.tlb`). Then use File | Save As to generate an IDL file with the same name. You'll have two windows to close after that.
3. Start EasyCOM2INC and select the TLB file. Click Start to generate the interface, and then Save to save an .INC file with the same name as the TLB that contains the interface declarations.
4. Click Class Generator to generate a class for the object. You'll see 2 listboxes in the window that pops up. If you see a "D" in the second column of the bottom listbox, click the big Toggle 'D' State button to turn it off. The 'D' signifies late-binding, which will perform even worse than the OLE control. It's on by default for certain types of files.
5. Exit the tool, and you're done.

This generates `Interop5.inc`, `Interop05_class.clw`, `Interop05_class.inc` and a bunch of other files you don't need to worry about. Now you need to add some items to your source code and to your project to make use of the new code.

1. Open your APP or PRJ using Project | Edit in the IDE. Click on the root folder and click Properties. Go to the Defines tab and paste in these magic defines:

```
_SVDllMode_=>0
_SVLinkMode_=>1
```

2. Somewhere in the global data section of your program, paste in this line (something like it

might already be there).

```
COMIniter CCOMIniter
```

3. Add an `INCLUDE` to put the class .INC file generated above into your project using whatever method you normally use (typically a global embed). In the example code, that's something like this:

```
program
include('Keycodes.clw')
include('Interop05_class.inc')
map
```

It's not necessary to put in the baseline `.INC` file as it's included in this one, and the `.CLW` file is compiled in based on a `MODULE` statement.

4. If you try to build this project, you'll find you don't have some common definitions that are in `ecom2inc.def`. You can find the file in the EasyCom2Inc install folder. Just copy it to some location where your `RED` file can find it, or update your `RED` file to point to it.

Now you can build and run the project just to be sure that you have all the source.

In the class `.CLW` file, you'll find a class named with an underscore, followed by the classname from your C# project, followed by _Class (e.g. `_Interop05ClassClass)`. You can just declare it in your data section with whatever scope is appropriate like any other class:

```
Interop05 _Interop05ClassClass
```

In the sample program I put it in the global section, but you will hardly ever do that in practice.

Before you use the class, you have to call the `.Init` method to get it ready to rock, and all return values are actually returned using a by-reference extra parameter. For example, the example method takes a `BSTRING` and returns a `BSTRING`, but the wrapper takes a `BSTRING` for the parameter, and an extra `*BSTRING` to hold the result. The wrapper returns a status instead of the result. The complete call syntax for this method is something like this:

```
Interop05.Init()
SourceBString = 'Test String With ' & Counter
assert(Interop05.ToString(COMResult) = S_OK, 'Failed in native COM call')
```

Using this generated wrapper takes about three times as much time as calling a native Clarion procedure, although to be fair quite a bit of that time is just generating the `BSTRING` to pass to .NET. On my machine, the whole 10,000 iteration loop only takes 30 ms, and my users can usually stand to burn 30 ms. If you really needed lightning speed, you could also remove some of the safety checks on the generated method and make it even faster. All in all, this is a really good interface, and very painless to generate and

use. It's my overall favorite method at the moment.

Native C++ Wrapper

The Visual C++ team created some magic code called IJW (for It Just Works) that allows you to create an ordinary DLL interface that can talk to .NET. Most of the marshaling and data conversion are handled by magic in the C++ language itself, and that which can't be handled that way can be done using a public `Marshal` class (passing strings back is sort of problematic). This interface is lightning fast, and extremely flexible. My test procedure that takes and returns a string takes only twice as long as the native Clarion procedure, and the whole loop only takes 20 ms for 10,000 iterations. It also has the advantage of making your .NET assembly accessible to any language that supports a C or PASCAL style interface.

I'm not going to cover how to do the C++ wrapper in detail here, but there's a sample module you can use as a framework, and if there's sufficient demand I'll write a future article on it.

Test Program

Interop05 is a program that shows passing a string to and from .NET using each of the available techniques. It performs the test 10,000 times for each type and displays the results. You can see that the OLE control is the slowest of the lot, but it doesn't fare all that badly.

Keep in mind that if you get into a situation where the bandwidth of this interface is a bottleneck, it's usually (but not always) a design problem. You should usually be doing the bulk of whatever you're doing on one side of the interface or the other.

Here are the results of the test. Keep in mind that as usual with benchmarks, these times are approximate at best.

Result	Time
Raw Loop. Just assigns to Clarion variable	0.000
Call Clarion Procedure	0.010
Call .NET Using Native C++	0.020
Native COM using EasyCOM2INC Wrapper	0.030
Call .NET Using OLE Control	0.260

Making life easier

Once you have the basic idea, there are a few things you can do to make your .NET experience easier:

1. If you possibly can, use Visual Studio 2005. Everything about it is much better than VS 2003.
2. Set up Visual Studio to run your executable for debugging.
    a. If you have a multi-project solution, pick one at random as the Startup Project. You can tell which project is the startup project because it's bold. Right click that project in Solution Explorer, and execute Project Properties.
    b. Go to the Debug tab, and click on Start External Program. Put in the fully qualified pathname for your Clarion executable. That way you can make changes to the .NET project, hit `F5` and it will build your .NET project and if it succeeds it will start the Clarion executable.
    c. Note that this won't work with Visual Studio 2003, but you're upgrading anyway... right?
3. Use the .NET debugger. If you've never used a really good debugger you're in for a pleasant surprise. If you've never used anything except the Clarion debugger you're in for a life-altering experience. To get started, put the cursor on a line you're interested in, hit `F9` to put a break point there, and hit `F5` to start up the Clarion program.
4. C# has a great documentation feature called XML Comments. Read about them in MSDN and use them.
5. Learn the .NET framework. It's the best part of .NET. There are hundreds of classes that do all kinds of cool things.
6. If you aren't reading blogs yet, get a good blog reader, such as the free SharpReader, and subscribe to some blogs. These will get you started:
   Code Project
   CodeGuru
   The Daily Grind
   Clarion News
   Clarion Articles

Gotchas

Here are a few quick tips to deal with problems you'll almost certainly run into sooner or later.

- If you're going to make any interface changes, unregister your assembly *before* you do the build to be sure your registry is reasonably clean. Use the `regasm` utility mentioned earlier with the `/u` parameter. Failure to do so tends to result in unpredictable behavior. You could even make a good argument for doing this every build in a pre-build event.
- You may find occasionally that you've accidentally put your assembly in the GAC. This usually happens to me when I get in a hurry. To find out if that's happened, just remove it. Run `GacUtil /u AssemblyName`. Note that you use the assembly name *without* the .DLL extension or path. If it's there, you'll see a message indicating it was changed. Note that I was serious about the *without* above. If you give GacUtil an actual filename, the utility will act as if it isn't there, even though it may be.
- If you think your registry might be corrupted, open `RegEdit` and search for every instance of your namespace. If you find it in a key with a GUID above it, navigate back to the GUID and delete the

whole key. Then register your assembly with `regasm` or build it again.

OK, let's do something fun

Now that you know how to pass any kind of data your heart desires back and forth between .NET and Clarion, I'll finish the series off with some fun stuff. This is not anything really finished... just something to get you off on a tangent, and maybe get some better cosmetics for your Clarion projects.

Let's start with a normal, somewhat dated looking Clarion window, like Figure 1.

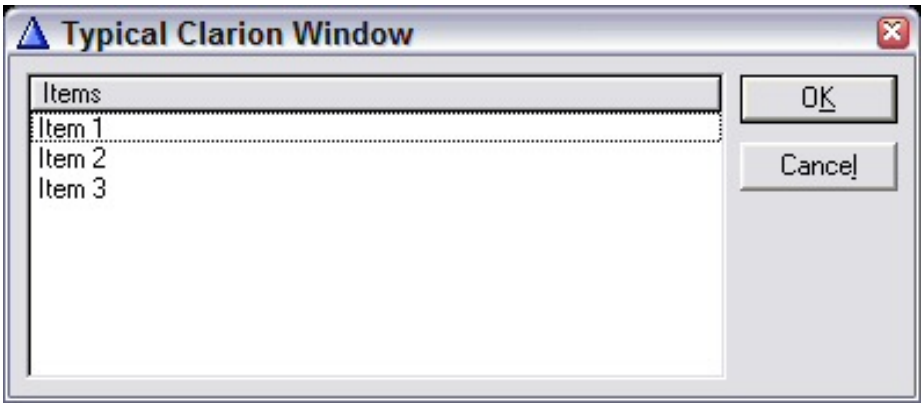


Figure 1. A typical Clarion window

The first thing to do is create an `EXP` file with the same name as the `PRJ` or `APP` file and put the word `MANIFEST` in it. This will enable Windows XP themes, which helps out the buttons a bit (Figure 2).

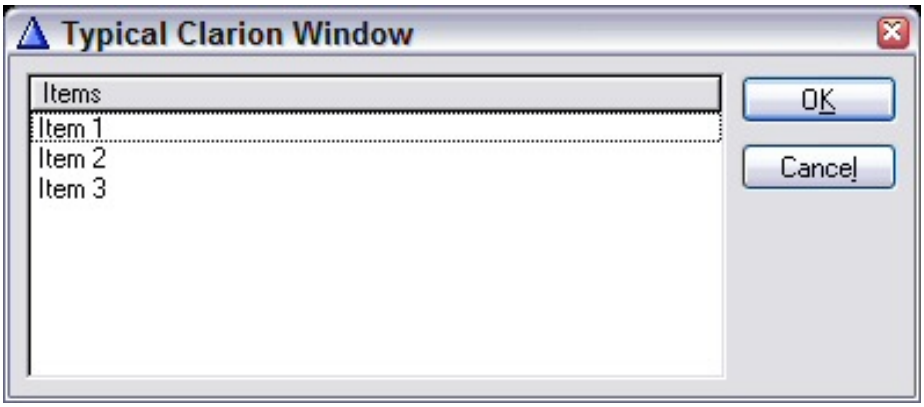


Figure 2. XP themes on buttons

Next, get rid of the `GRAY` attribute and make a background color, or maybe use an image. For an example, I'll use the standard HTML color Khaki. If you want to play around with colors, take a look at my HTML Colors chart for a good place to start. Changing the background unfortunately changes the listbox at the same time, so you'll have to give the listbox a color attribute as well. This gives you something like Figure 3, which obviously needs some more work but it's a step in some direction.

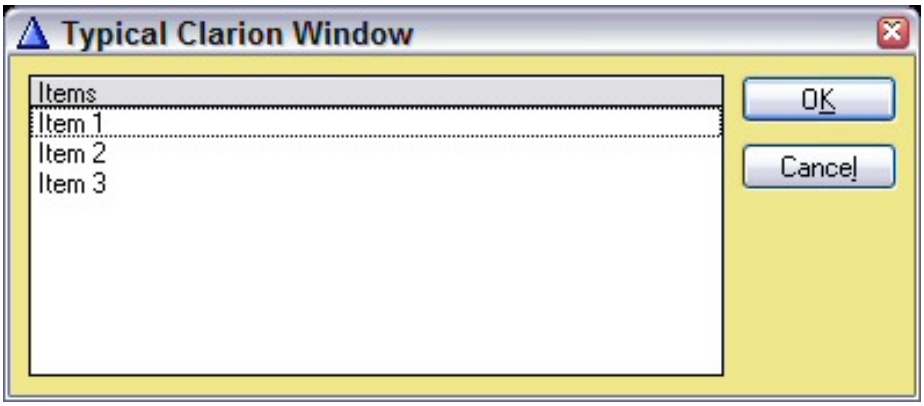


Figure 3. Adding background colors

Everything I've done so far is ordinary Clarion code. I did it using evil hard-coded window declaration stuff instead of in a library or base-class, but it's all pretty ordinary stuff. Now, I'll show a C# class using the same techniques I used before that exposes a method called `AttachDropShadow`. I call it using the same techniques from way back in segment one with this code:

```
Lipstick = CREATE(0, CREATE:OLE)
Lipstick{PROP:Create} = 'Interop07.Lipstick'
Lipstick{'AttachDropShadow(' & ?List1{PROP:Handle} & ')'}
```

Here I'm using Clarion's property syntax to pass the `hWnd` of the listbox to the method. Within that method, I derive from the handy `NativeWindow` class, which allows me to easily wrap functionality around a window. This class takes care of all the nasty details of subclassing and managing scope and lifetime. You just create a new object that's derived from `NativeWindow`, use the `AssignHandle` method to attach it to a window, create a `WndProc` virtual method and viola... you can now do subclassed redrawing in managed code, using GDI+. Trust me on this one, this is very cool and greatly expands what you can do.

For a relatively trivial example, I added a drop shadow to the listbox. This is admittedly not the most attractive drop-shadow ever invented, but it's all done in about 20 lines, and you can debug it by single-stepping through the paint events. Now you have some real power at your fingertips. Figure 4 shows the same window with a basic drop-shadow.

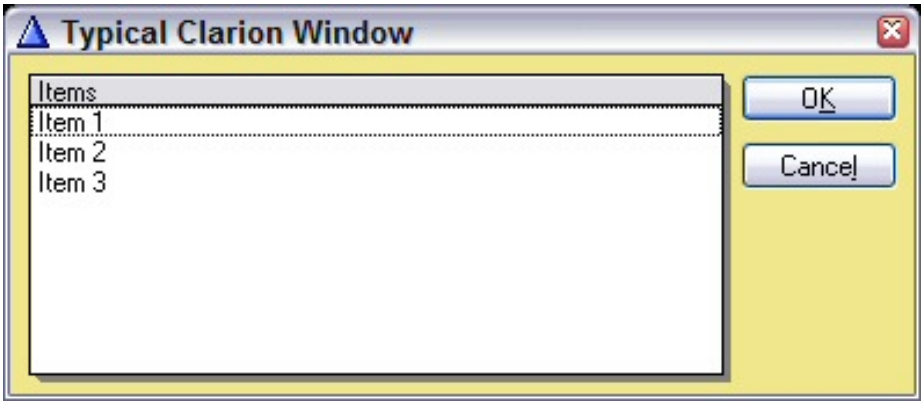


Figure 4. Adding a drop shadow

Once you have this idea, you can do lots of things quite easily that were very difficult in Clarion. If you want to round the corners, or change the gray in the listbox to another color, or make an entire customizable skin system, you can do it. Do it properly in your base classes and you can apply it to existing projects. In fact, I'm doing that right now for my company's very large Clarion application.

If you examine the source carefully, you'll see that I'm mixing ordinary API calls with GDI+ calls. This is often necessary when doing graphics programming in .NET. The .NET framework is very large, but not as big as the Windows API.

Conclusion

In this series, I've shown you how to mix-and-match Clarion and .NET in a number of ways, and a bit of a teaser for how to update the cosmetics for Clarion applications. The .NET framework is really large, well-supported and growing. The .NET languages themselves aren't necessarily that much better than Clarion is, but the available toolset is much bigger. With the techniques in this series, you don't have to choose one over the other. You can still use Clarion right here and now, and still enjoy the benefits of the .NET framework. You could even use Clarion's AppGen to generate .NET code if you really wanted to... but that's a story for another day.

Download the source

Wade Hatler has been around Clarion so long he got the very first patch release for Clarion 1.0 for DOS. That was back when Clarion had a dongle, no compiler and no templates. Wade co-owned IntelliScan, a company producing third-party Clarion products for several years. For the last 10 years he has worked for IMPAC Medical Systems creating software for the management of Cancer Therapy. He recently completed a 3-year, 12,000 mile, 12 country bicycle tour of the world, during which he carried a laptop and worked about half-time. He lives with his wife and daughter near Seattle. You can reach Wade at *WadeHatler@wademan.com*.

Reader Comments

Add a comment

- » May 2006 MSDN article explains how to call .Net from VB 6...

Clarion Magazine

Updating Hot Fields

by Steven Parker

Published 2006-04-11

On a browse window, Hot Fields significantly extend my ability to display information. Fields for which there is no space in the list control, data from other files (related or not), run time computed data are all nicely handled with Hot Fields.

In most cases, I use string controls to display Hot Fields. With the "b" switch added to the display picture (to blank empty fields), a clean look is easy to achieve, as show in Figure 1.

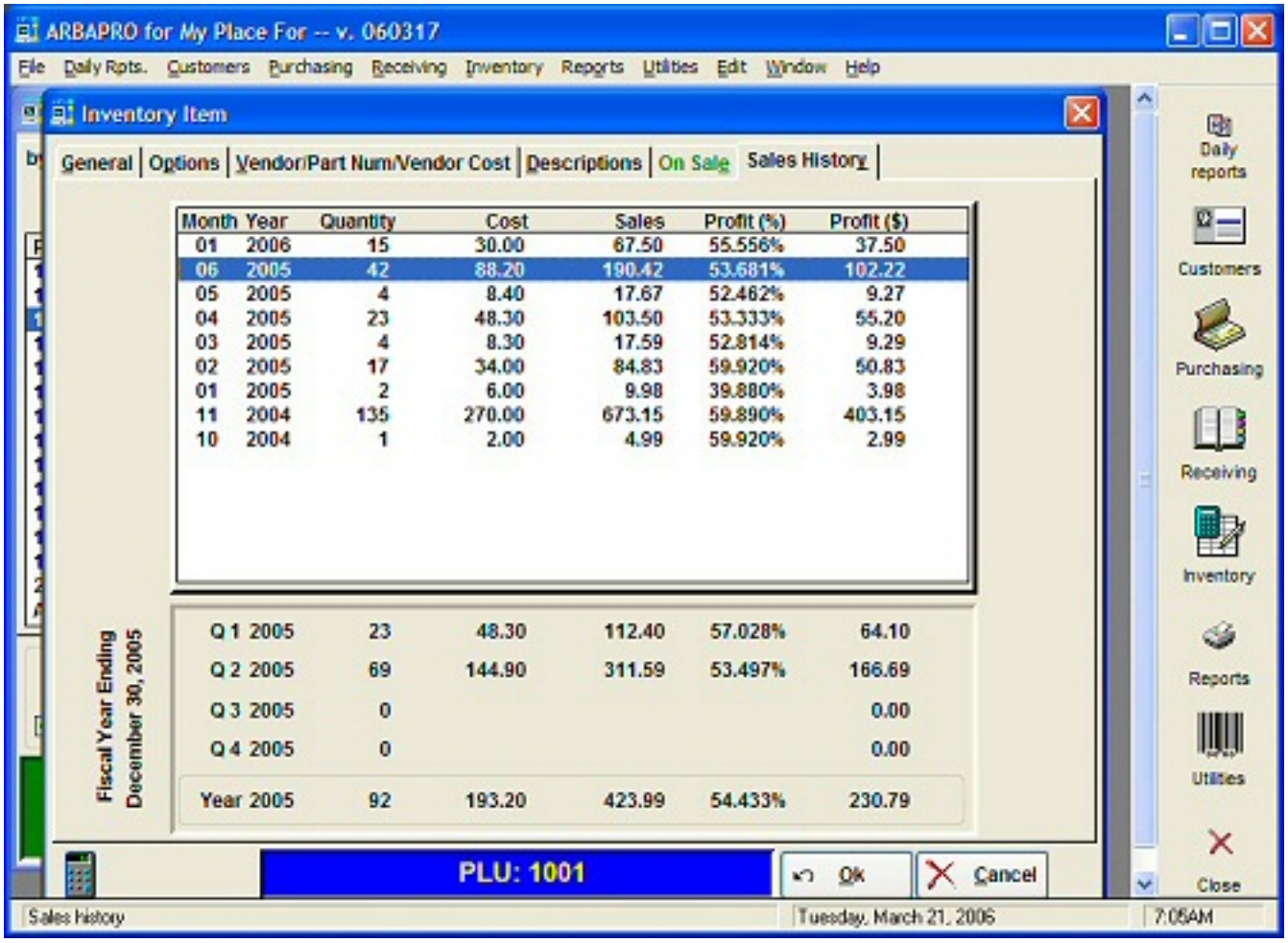


Figure 1. Hot Field displayed in strings (view full size image)

Then I made a fatal mistake. I populated a set of Hot fields using Entry controls. On rare previous occasions, I have used entries but I also set those fields' backgrounds to `COLOR:BTNFace`. This time, I forgot.

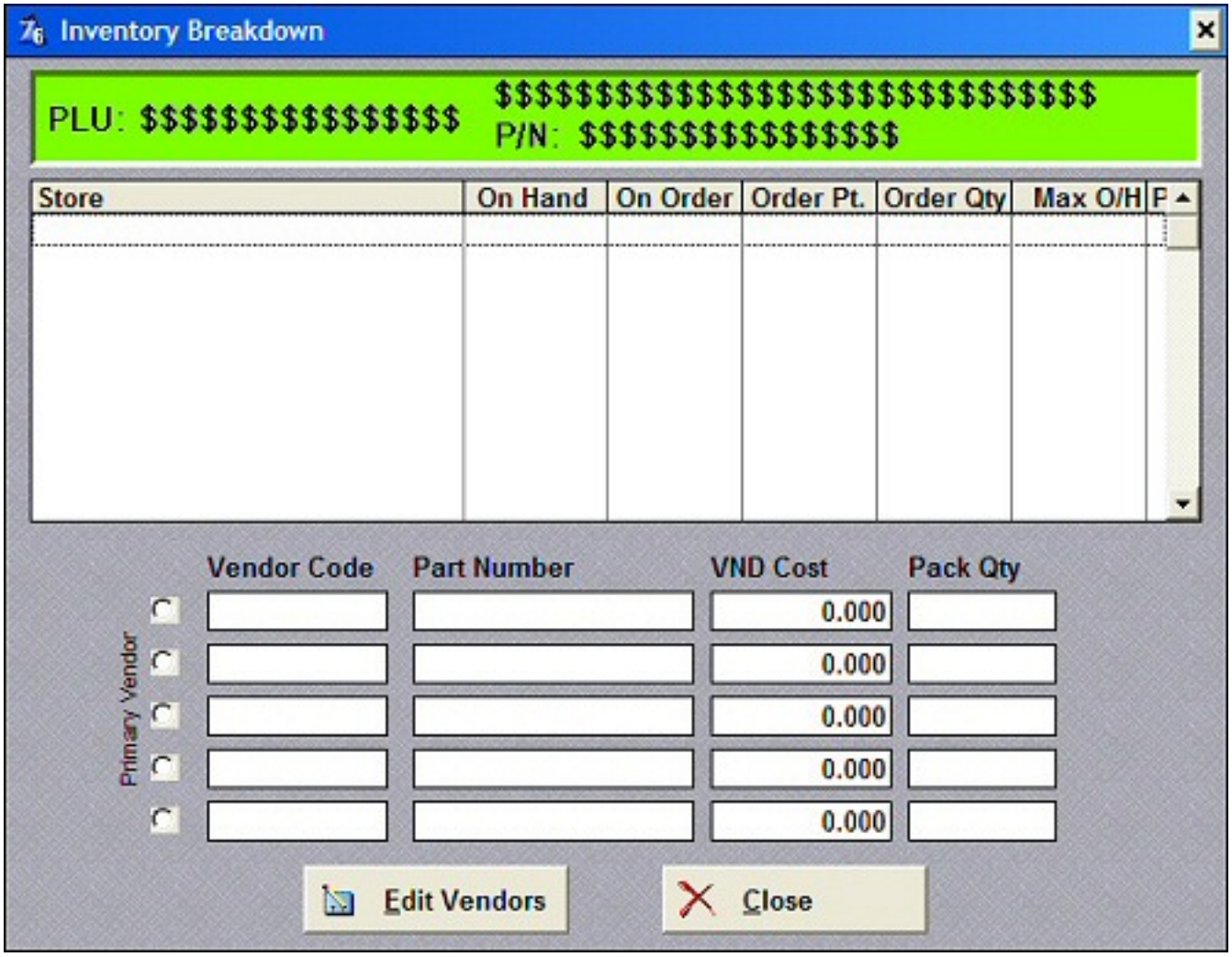


Figure 2. Hot Fields in Entry controls (view full size image)

I was trying to get a look appropriate to the context. The browse in Figure 1 displays computed data and users know it. While they might wish they could change the numbers, they know that they can't, at least within a computer program. But the Hot Fields in Figure 2 are real file fields. They contain information that a user might very well want to edit.

The fact that the user might want to edit one of them is the very reason they *are* displayed. As a user scrolls through the list, they can decide which part number(s) or cost(s) needs editing. They might want to add a vendor for one of the stores. Of course, each entry box is set for Skip and Read Only. These fields, in other words, *can't* be edited.

That, as the reader might suspect, does not stop users from clicking in the control and attempting to edit it. While the Edit Vendors button calls an update form on which the user can change the record to their heart's content, they continue attempting to edit the Hot Fields directly on the browse. Being read only, they can't and resolve their dilemma by calling in a bug report.

The horse having been let out of the barn, there is no use trying to close the barn door. I can't switch to strings. Neither can I change the fields' backgrounds. Thus, the current exercise: How can I edit Hot Fields on a browse?

Nothing I know about the Clarion language, nothing I know about browses immediately suggest a way of accomplishing what is wanted. So I fell back on that tried and true development tool, trial and error.

Trials and errors

I started by removing the Skip and Read Only attributes from each of the fields. In the attached test application (created in 5.5), see the first menu item (Inventory (Normal Hot Fields)) under Inventory. I can edit any of the Hot Fields and, as one would expect, nothing is saved when I scroll to another inventory item or close the browse.

Strike one. But no surprise.

Next, I populated a SaveButton control template. It is described as the template to Write Records to a data file. It is the key control template on a Form. In the test app, try the Inventory (Save Button) item. No errors are thrown. This is good.

Neither, however, is anything saved. This is not so good.

Strike two. Big surprise. I thought that saving is just what this control template is supposed to do.

What now? I replaced the `SaveButton` button with a plain button and embedded:

```
PUT(Inventory)
If ErrorCode()
  Message(ErrorCode() & ': ' & Error(),'Error',ICON:Hand)
End
```

Not very "ABC" but I do know that `PUT()` will give me an `ErrorCode()` and an `Error()` that should (famous last words) tell me something useful.

Try the Inventory (Force Save) menu item in the test app. It does give a useful, very useful diagnosis of what is going on: the dreaded "Record Not Available" message.

Record Not Available and browse architecture

As soon as I saw that error, I realized precisely why records weren't updating. Worse, I realized that I had written a number of articles, some of them about Clarion DOS, explaining the underlying issues.

First, and the immediate cause of this error, file I/O statements (`Add`, `Put`, `Delete` and their ABC wrapper methods) make certain assumptions about the file buffer. `Add` assumes that it started with an empty buffer; `Put` and `Delete` assume that the buffer contains the record to be affected.

In fact, the file buffer for my `Put` does not contain the record highlighted in the browse. In fact, the file buffer is empty. Hence the error 33. Critical assumption violation.

The fact that the buffer is empty is due to the second vital fact: browses do not fill the file buffer.

Typically the buffer is filled by a `Get` or its ABC wrapper. Highlighting a browse record issues no `Get` or `Fetch`. Therefore the buffer is empty.

I remember writing about it: a browse reads a disk file and creates a view. It uses this view to fill a queue (with just enough entries to fill the list control). The list box displays the queue.

Nowhere is a specific record read. (To be precise, when a browse update button or Select button is pressed, a record *is* read. But that fact is not germane here.)

Planting palm firmly on forehead

Major "doh!" moment here. I need to get the record corresponding to the highlighted queue (browse) record. And, not only do I know how to do that, I know two ways of doing it.

First, I can retrieve the underlying record as I scroll through the browse. This is similar to the DOS option, Enable Hot Records. To do this, in TakeNewSelection, After Parent call embed:

```
Get( file , key)
```

or

```
Access: file . Fetch( key )
```

This works because I have ensured that each node of the key is in the view (either populated in the list box or added to the Hot Fields list). So, for example, `INV:PLU` has a value on each row of the browse and that same field is the only member of the `INV:PLUKey`. Et violá.

The other method is to populate a button, like my Edit Vendors button. When that button is pressed, I do the record fetch.

The first method will slow browsing down. How much slower depends on network speed, number of records, filters, etc.

The second method requires a user action. It's your decision.

One solution

I opted for the make-the-user-press-a-button method. See the test app, Inventory – One Solution.

But I took things a step further.

To prevent users from "accidentally" clicking on one of these fields and thinking they could edit, I placed a Region control over the fields:

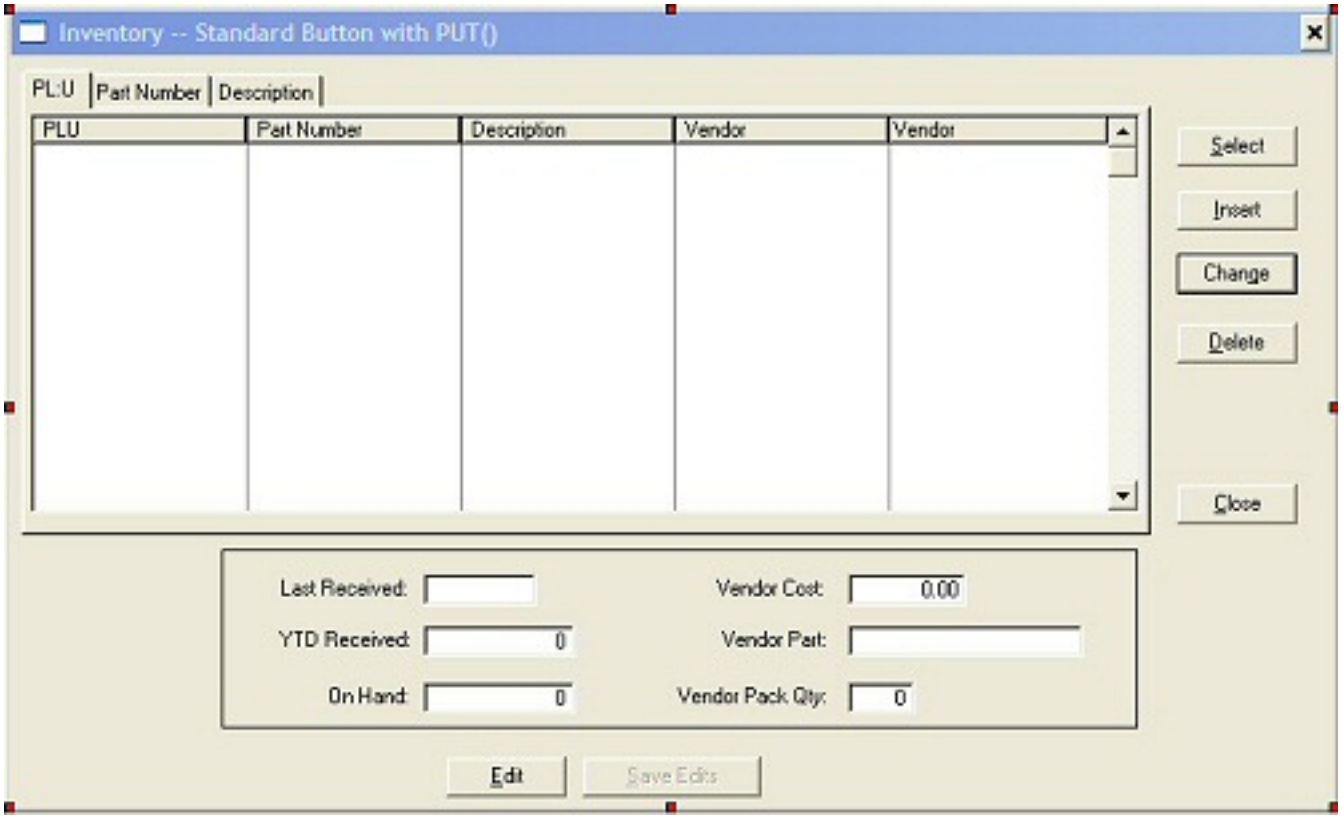


Figure 3. Window in design mode (view full size image)

This makes the fields non-selectable. I also disable the Save Edits button.

The fun starts when the user clicks the Edit button:

```
?Region1{Prop:Hide} = True
?Button6{Prop:Disable} = False
Access:Inventory.Fetch(INV:PLUKey)
```

I hide the region. The fields thus become selectable. Then I enable the Save button and, finally, I get the record.

When the user presses the Save button:

```
PUT(Inventory)
If ErrorCode()
  Message(ErrorCode() & ': ' & Error(),'Error',ICON:Hand)
End
Select(?Browse:1)
?Region1{Prop:Hide} = False
?Button6{Prop:Disable} = True
ThisWindow.Reset(1)
```

I write the record and test for errors. I move focus back to the list and reset the Region and Save button.

The window reset ensures that the new data display correctly when the user scrolls up and down. `ThisWindow.Reset` may be a bit much; `ResetFromFile` or `ResetQueue` might do just as well. (Try commenting this line out, make some edits, then scroll about. Then, check the data using Topscan.)

You may note that the test app is incomplete. You can make a change, scroll to another record without pressing Save and the buttons do not reset.

Summary

Who'd'a thunk it? Hot Fields on a browse are editable! One must remember, of course, that one is responsible for the record buffer (so, if the primary key has more than one node, make certain all are primed before retrieving the record). With that in hand, the rest is easy.

But what I'd really like is a "wayback machine" so I can go back and populate those Hot Fields as strings.

Download the source

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

Reader Comments

Add a comment

- » Hi Steve i think this article is for beginners. we look...

Clarion Magazine

Multi-User Primary Keys: A Solution

by Rhys Daniell

Published 2006-04-07

It's easy to forget that back in the early days of Clarion, most applications had a single user. These days it's not uncommon to find hundreds of Clarion end users operating over a single database. Clarion has scaled amazingly well, except for one niggling flaw. This flaw isn't Clarion's fault per se; it stems from the table/form paradigm so loved by users and developers, and it arises when you have multiple users using the same form to add records to a table with an incrementing primary key.

There are other kinds of primary keys, but a numeric primary key (often called a SysID) which cannot be modified by the user is increasingly desirable and brings benefits when working with SQL and development tools including Clarion.

The easiest way of generating primary key values is to use Clarion's autoincrement feature. When you open a form to add a record, this finds the next available value for the primary key, and then reserves it for you by adding a record (mostly blank except for the primary key value) to the table. If you cancel out of the form, the record is deleted.

This brings drawbacks if you have multiple users accessing the same table/form simultaneously. At the very least, if a user views the browse while another user is adding the record, they will probably see what appears to be a blank record in the browse list. Depending on your database design, the second user may even get a duplicate key error message if they try to insert a record at the same time as the first user. If the second user completes an add, and the first user subsequently cancels theirs, there will be a gap in the primary key sequence. This *shouldn't* matter - after all the primary key is supposed to be hidden from the end user - but it *does* matter a lot to a small handful of obsessive users!

Finally, if a user suffers a system crash while the insert form is open, a blank record will remain in the browse until deleted. If users are in the habit of deleting such blanks, sooner or later they will delete a record which is a legitimate add in progress, causing another user to lose their work. In our case, the last straw was a ClarioNET implementation of an application where the remote operators (perversely or otherwise) took to shutting down the ClarioNET client without leaving the form, leaving large numbers of orphan records.

The most obvious solution to this problem, provided you are using a SQL back end, is to use an `IDENTITY` field for the primary key. This will cause the database to generate a unique value for the primary key when the record is added, taking away the need to "reserve" the value at the start of the add. This is a good solution, unless you plan to add child records within your form, in which case you need the primary key value to establish the record relationships (and have Clarion's referential integrity delete the child records when you cancel the add).

A solution

Well, here's another solution. While it's based on using a SQL database, it could be adapted to other file systems. The key

points are:

1. Create a managed "stack" of primary IDs.
2. At the start of an add, get an ID from the stack, and reserve it.
3. If the add is cancelled, make the ID available again.
4. If the add is completed, use the ID and mark it non-available.

You will need a `PrimaryKeys` table which keeps track of available IDs for any number of tables. Fields include:

Field	Type
TableName	VARCHAR(30)
PrimaryID	INT
Status	TINYINT
ReservedDateTime	DATETIME

Initially, this table stores the last used `PrimaryID` for each table. The `Status` is set to 1 (in use).

When an insert form is opened, a `GetNextPrimaryID()` function adds a new record with the next `PrimaryID` and a status of 0 (reserved). (That's just the simplified version - see below for more.) When the insert form is completed, an `UpdatePrimaryID()` function sets the status of this record to 1 (in use).

If the insert form is cancelled, `UpdatePrimaryID()` sets the status to 2 (discarded).If the record is subsequently deleted by the user, `UpdatePrimaryID()` sets the status to 3 (deleted).

Now, what happens if the user opens the form and never comes back (due to a system crash, or worse)? Even though the status is set to 0 (in use), `GetPrimaryID()` assumes it's available if more than 24 hours old (expired). IDs for discarded and deleted records are assumed to be immediately available. In other words, primary key values which are not used are recycled, eliminating those holes in the number sequence. And users can add as many records simultaneously as the hardware allows, without seeing blank records in the browse.

Source code for all the above is available below. It's written for MSSQL and the Clarion (legacy) templates, but the same principles can be applied to other database systems and the ABC templates. Please note that some coding skills are required to get it working in your application - anyone (including SV) who wants to turn it into something plug 'n' play is welcome to do so.

Here are the steps you need to take:

1. Create the `PrimaryKeys` table (SQL script below)
2. Create the `GetNextPrimaryID` and `UpdatePrimaryID` SQL stored procedures
3. If you have existing data, add starting values to `PrimaryKeys` table
4. Create a Clarion source procedure to call the stored procedures
5. For each table with an autoinc primary key, remove the autoinc attribute in the Clarion dictionary
6. For each table with an `IDENTITY` primary key, modify the SQL database to make this field non-identity
7. Add the extension template to the form for each table.

Step 1 – Create the PrimaryKeys table

Here's the SQL script to create the PrimaryKeys table and accompanying index (written for MS SQL):

```
CREATE TABLE PrimaryKeys(
   TableName    VARCHAR(30),
   PrimaryID    INT NOT NULL,
   Status       TINYINT DEFAULT(0),
   ReservedDateTime DATETIME DEFAULT(GetDate()),
   CONSTRAINT KTableNamePrimaryID
     PRIMARY KEY(TableName, PrimaryID))
GO
CREATE UNIQUE INDEX KTableNamePrimaryIDStatus ON
  PrimaryKeys(TableName, PrimaryID, Status)
GO
```

Step 2 - Create two stored procedures

These two stored procedures reside on the MS SQL server.

```
CREATE PROC GetNextPrimaryID @TableName CHAR(30)
   AS
 DECLARE @NextPrimaryID  INT
  -- see if there's an expired value
   SELECT @NextPrimaryID = MIN(PrimaryID)
     FROM PrimaryKeys
     WHERE TableName = @TableName
     AND (Status > 1 OR (Status = 0
         AND ReservedDateTime < GetDate() - 1))

    IF @NextPrimaryID IS NOT NULL    -- found a disused one
    BEGIN
       UPDATE PrimaryKeys
         SET ReservedDateTime = GetDate(), Status = 0
         WHERE TableName = @TableName
                 AND PrimaryID = @NextPrimaryID
         GOTO EndProc
    END
  -- get the next value
   SELECT @NextPrimaryID = MAX(PrimaryID) FROM PrimaryKeys
     WHERE TableName = @TableName
   IF @NextPrimaryID IS NULL SET @NextPrimaryID = 0
   SET @NextPrimaryID = @NextPrimaryID + 1
   INSERT INTO PrimaryKeys(TableName, PrimaryID)
     VALUES(@TableName, @NextPrimaryID) -- other values set by default
EndProc:
   SELECT @NextPrimaryID
GO
CREATE PROC UpdatePrimaryID @TableName CHAR(30),
  @PrimaryID INT, @Status TINYINT
   AS
-- Status:   0 = reserved 1 = taken 2 = discarded 3 = deleted
 UPDATE PrimaryKeys
   SET Status = @Status
   WHERE TableName = @TableName AND PrimaryID = @PrimaryID
```

```
GO
```

Step 3 - Add starting values

You only need to prime the PrimaryKeys table if you already have data in the table which is getting its IDs from `PrimaryKeys`. Replace `'Customers'` and `'CustID'` with the name of your table and its primary key.

```
INSERT INTO PrimaryKeys(TableName, PrimaryID, Status)
  SELECT 'Customers', MAX(CustID), 1 FROM Customers
```

Step 4 - Create a Clarion source procedure

You'll need this Clarion source procedure to call the stored procedures. In this example, `ResultsTable` is a special table I use to get results from a SQL query. You can use *any* table which has a `LONG` as the first field.

These are the procedure prototypes (you won't need to add these separately if you're create the source procedure within an APP):

```
GetPrimaryID(STRING TableName, *LONG PrimaryID), BYTE
UpdatePrimaryID(STRING Tablename, LONG PrimaryID, |
                BYTE NewStatus)
```

And here are the procedure definitions:

```
GetPrimaryID   FUNCTION (pTableName, pPrimaryID)
! gets the next primary ID off the 'stack'
! (PrimaryKeys table) and reserves it for 24 hours
FilesOpened     BYTE(0)
RetVal          BYTE(0)
  CODE
  DO OpenFiles
  ResultsTable{Prop:SQL} = 'CALL GetNextPrimaryID(''' |
    & CLIP(pTableName) & ''')'
  IF ERRORCODE()
     MESSAGE('Database error generating primary id -|' & |
       ERROR() & '|' & FILEERROR(), 'System Error', Icon:Hand)
     DO ProcedureReturn
  .
  NEXT(ResultsTable)
  pPrimaryID = RT:Field1
  IF pPrimaryID THEN RetVal = TRUE.
  DO ProcedureReturn
  DO CloseFiles
  RETURN(RetVal)

OpenFiles  ROUTINE
  PUSHBIND
  CheckOpen(ResultsTable,0)
  FilesOpened = True

CloseFiles ROUTINE
  POPBIND
```

```
UpdatePrimaryID      PROCEDURE (pTablename, pPrimaryID, |
                                pNewStatus)
! Start of "Data Section"
! [Priority 3500]
! Status:   0 = reserved  (made available in 24 hours
!                          if not selected)
!           1 = taken     (not available any more)
!           2 = discarded (form add cancelled, now available)
!           3 = deleted   (record deleted, now available)
!                          I guess reuse could be made optional
!
! most importantly, to prevent a PrimaryID being
! reused, set status = 1

FilesOpened     BYTE(0)

  CODE
  DO OpenFiles
  ResultsTable{Prop:SQL} = 'CALL UpdatePrimaryID(''' & |
     CLIP(pTableName) & ''', ''' & pPrimaryID & ''', ''' |
     & pNewStatus & ''')'
  IF ERRORCODE()
     MESSAGE('Database error updating primary id -|' & |
       ERROR() & '|' & FILEERROR(), 'System Error', |
       Icon:Hand)
  .
  DO CloseFiles
OpenFiles  ROUTINE
  PUSHBIND
  CheckOpen(ResultsTable,0)
  FilesOpened = True
CloseFiles ROUTINE
  POPBIND
```

Step 5 - Remove autoinc

You must remove the autoinc attribute for your table's primary key in the Clarion dictionary;

Step 6. Remove IDENTITY

You also need to make sure there's no `IDENTITY` attribute on the primary ID field in your table. Ideally, you should make a copy of the table, drop and recreate it, and re-import the data. Alternatively, create a temporary column to hold the current primary key value and then drop and recreate the primary key column without the `IDENTITY` attribute.

Step 7 – Add the extension template

You'll need to add the following extension template to any forms that use this autonumbering technique. Note that this extension is written for the Clarion (legacy) templates and may need changes to work with ABC. Put the following code in a TPW file and add an appropriate `#INCLUDE` statement in your TPL file.

```
#! Templates, Clarion code, and SQL script for
#! 'Multiuser primary keys' ClarionMag article
#! Author: Rhys Daniell, rhys@capebyronsystems.com.au
#|
#!  author takes no responsibility (attribution would be nice)
#!  you are welcome to re-use, modify, and re-distribute
#!
#EXTENSION (GetPrimary, 'Get next primary key value'), PROCEDURE, FIRST
#PROMPT('Table requiring primary key value:',FILE),%PrimaryFile, DEFAULT(%Primary),
REQ
#DISPLAY('Gets next available primary key value from ''stack'' of values ')
#DISPLAY('Requires GetNextPrimaryID and UpdatePrimaryID')
#DISPLAY(' functions (which use stored procs of same name)')
#DISPLAY(' and PrimaryKeys table')
#DISPLAY('')
#DISPLAY('(c) Cape Byron Systems 2006')
#AT (%PrimeFields), FIRST
#FIX(%File, %PrimaryFile)
#FIX(%Key, %FilePrimaryKey)
#SELECT(%KeyField, 1)
IF ~GetPrimaryID('%PrimaryFile', %KeyField) THEN DO ProcedureReturn.
#ENDAT
#AT (%BeforeFileClose)
#FIX(%File, %PrimaryFile)
#FIX(%Key, %FilePrimaryKey)
#SELECT(%KeyField, 1)
IF OriginalRequest = InsertRecord
   IF LocalResponse = RequestCompleted
      UpdatePrimaryID('%PrimaryFile', %KeyField, 1)    ! 1 = taken
   ELSE
      UpdatePrimaryID('%PrimaryFile', %KeyField, 2)    ! 2 = discarded
      IF RIDelete:%PrimaryFile().
   .
ELSIF OriginalRequest = DeleteRecord AND LocalResponse = RequestCompleted
   UpdatePrimaryID('%PrimaryFile', %KeyField, 3)       ! 3 = deleted
.
#ENDAT
```

You are welcome to use this code as you see fit. Enjoy!

Rhys Daniell never meant to become a programmer but in 1988 CPD turned out to be the best way to solve a business problem and the world began to beat a path to his door. Getting the Clarion religion included running Australian Devcons and (for a time) the Sydney User Group. These days Rhys and a small team are comfortably located in the rolling green hills of Northern New South Wales, servicing customers around Australia in the security, business services franchise, and agricultural industries.

Reader Comments

Add a comment

- » Here are the changes I made to get this to work for...

Clarion Magazine

The ClarionMag Blog

Get automatic notification of new items! RSS feeds are available for:

XML All blog entries
XML All new items, including blogs

Blog Categories

- »All Blog Entries
- »Clarion 7 Clarion.NET
- »Future Articles
- »News flashes
- »Nifty Stuff

Visual Studio Express stays free

Direct link

Posted Thursday, April 20, 2006 by Dave Harms

I blogged back in November about Microsoft's decision to offer their Visual Studio Express products (Web Developer, SQL Server, Visual Basic, C#, Visual C++, and Visual

J#) free for one year (meaning download them during that year, and you can keep them forever). Apparently response has been pretty good (five million downloads) so MS has decided to keep the offer open, as noted by Dan Fernandez. The Express products will remain free.

There are a numer of other cool items in that blog post, including a .NET interface for the Lego Mindstorms Starter Kit and home automation using Skype.

9052 on the way

Direct link

Posted Monday, April 10, 2006 by Dave Harms

Bob Z has blogged that 9052 has been released to third party vendors for compatibility testing. The 9052 release includes some important bug fixes, such as a SUSPEND failure problem, embed editor error on using the Version template, and an XML parser GPF. See the complete list of changes.

Clarion 7 adds TPS encryption methods

Direct link

Posted Thursday, April 06, 2006 by Dave Harms

Bob Z has blogged about a new feature in the C7 TPS driver: developer-specified encryption techniques:

> With the introduction of Clarion 7.0, you will have the ability to use any encryption algorithm supported by any encryption provider that plugs into the Windows encryption subsystem. This will enable developers to create, store and exchange data in a very secure environment.

Interesting stuff, and I hope this presages more C7 news!

.NET, decompiling, and obfuscation

Direct link

Posted Monday, April 03, 2006 by Dave Harms

Jason S. posted a question in the softvelocity.clarion.chat newsgroup about decompiling .NET programs, and the more I thought about this the more I thought a more detailed treatment of the question is in order. Just not enough for an article<g>.

When you compile a .NET application, you're not creating a file that the OS can load directly; instead, you're creating a file (or files) containing, among other things, program code in Intermediate Language (IL) code. IL is a much higher level language than the executable code in a typical EXE, and it's a lot easier to reconstruct the original source from IL code than it is to reconstruct, say, the original C++ code from its compiled form. In IL, variable names, method names, and program logic are all preserved.

If protecting your source is important (and quite possibly it's not only important but essential) you'll want to garble any code you deliver to clients, using a tool called an obfuscator. Obfuscators typically change all variable names to meaningless values (e.g. `customerName` becomes `a`, `customerAddress` becomes `b`, etc.), and do the same for classes and methods. They may also make static data harder to read, perhaps by changing string encoding, or using functions to return static data. Obfuscators can also make program flow more difficult to follow, by rearranging code using goto statements, introducing never-called logic statements, inlining functions, reversing loops, and so forth. Obfuscators may also be designed to take advantage of known weaknesses in decompilers and deobfuscators.

For further reading, see this article on the Palisade security site.

Enterprise-strength obfuscators tend to have enterprise price tags. I have, however, come across the e-book Obfuscating .NET: Protecting Your Code from Prying Eyes by Dan Appleman. When you buy the book, you also get a download to an open source obfuscator, as explained in the book. I haven't read the book or tried the obfuscator, but it sounds like an option worth exploring.