

Clarion MAGAZINE

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

[View Recently Posted Reader Comments](#)

Okay, so being able to add your own comments to ClarionMag articles is cool. But how do you locate newly posted messages without visiting the same articles over and over? You visit this page!

Posted Thursday, May 31, 2001

[The Clarion Advisor: SUB Tricks](#)

You've probably used the SUB function to extract one string from another, but did you know that SUB can also count backwards?

Posted Thursday, May 31, 2001

[The Novice's Corner:](#)

[Understanding EQUATES.CLW \(Part 2\)](#)

In any programming environment it's useful to represent certain commonly-used values as constants. In Clarion, the EQUATE keyword defines such a constant, and the two places you'll find most of these equates are in EQUATES.CLW (naturally) and PROPERTY.CLW. In this second of two parts, Dave Harms finishes discussing EQUATES.CLW.

Posted Thursday, May 31, 2001

[Loading DLLs At Runtime - Part 3](#)

Do you want to sell your software with optional modules that are automatically recognized when installed? Do you ever need to call a procedure that may not exist on your end user's system? Will your program even load if you use one of those functions? In this three part series Larry Sand explains how to load DLLs at runtime.

Posted Tuesday, May 29, 2001

[Weekly PDF for May 21-27, 2001](#)

All Clarion Magazine articles for May 21-27, 2001.

Posted Monday, May 28, 2001

[File Explorer 1.7 Shipping](#)

[NetTalk 1.0 Beta 12 With DUN Available](#)

[CapeSoft Draw Version 1.0 Beta 2 Available](#)

[CapeSoft Mailer 1.0 Beta 1 Released](#)

[CapeSoft MessageBox Version 1.0 beta 1 Released](#)

[The CapeSoft Big Birthday Bash Ends Soon](#)

[One Week Left On FrameText Special](#)

[Search Engines Profile Exchange Updated](#)

[Buggy 2.1.4 Available](#)

[IMPEX 5.0 Adds HTML Export](#)

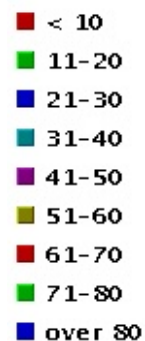
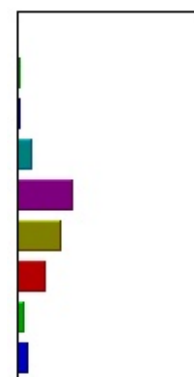
[FrameText Special Offer](#)

[Stealth Software Mail & Fax Upgrade Pricing Changes](#)

[Clarion To Excel](#)

SURVEY

On average, how many hours per week do you work?



Vote: 193

Creating Elliptical Windows in Clarion

Have you ever wondered how some applications display windows in non-standard shapes? As Brice Schagane shows, you can create elliptical windows (and other shapes) with just a few simple API calls.

Posted Friday, May 25, 2001

Loading DLLs At Runtime - Part 2

Do you want to sell your software with optional modules that are automatically recognized when installed? Do you ever need to call a procedure that may not exist on your end user's system? Will your program even load if you use one of those functions? In this three part series Larry Sand explains how to load DLLs at runtime.

Posted Tuesday, May 22, 2001

Weekly PDF for May 14-20, 2001

All Clarion Magazine articles for May 14-20, 2001.

Posted Monday, May 21, 2001

Reading Tables With ADO

Have you ever wanted to write a generalized utility to work with a data file which may exist on more than one backend database? Do you need a utility to handle a file when you don't have/or want a DCT layout? Have you ever wanted to use ADO (ActiveX Data Objects) in Clarion as a standard way of managing your data? Here's how to get started.

Posted Monday, May 21, 2001

The Novice's Corner: Understanding EQUATES.CLW (Part 1)

In any programming environment it's useful to represent certain commonly-used values as constants. In Clarion, the EQUATE keyword defines such a constant, and the two places you'll find most of these equates are in EQUATES.CLW (naturally) and PROPERTY.CLW. In this first of two parts, Dave Harms explores EQUATES.CLW.

Posted Friday, May 18, 2001

Using The TPS ODBC Driver

Vince Du Beau explore the possibilities of using the TPS ODBC driver with other applications, and demonstrates importing data into an Excel spreadsheet.

Example

**CapeSoft Draw
Version 1.0 Beta 1
Released**

**The CapeSoft Big
Birthday Bash**

Posted Thursday, May 17, 2001

Loading DLLs At Runtime - Part 1

Do you want to sell your software with optional modules that are automatically recognized when installed? Do you ever need to call a procedure that may not exist on your end user's system? Will your program even load if you use one of those functions? In this three part series Larry Sand explains how to load DLLs at runtime.

Posted Wednesday, May 16, 2001

Weekly PDF for May 7-13, 2001

All Clarion Magazine articles for May 7-13, 2001.

Posted Monday, May 14, 2001

The Clarion Advisor: Avoiding GPFs With ANYs And QUEUES

If you've every used an ANY variable in a QUEUE, chances are you've encountered at least one GPF. ANYs in QUEUES require some special handling, and unfortunately the Clarion documentation is not completely accurate.

Posted Thursday, May 10, 2001

Quickbooks-Style Date Fields

Andrew Guidroz II gets a request for QuickBooks-style date incrementing/decrementing, and writes a template to automatically apply this code to an entire application.

Posted Thursday, May 10, 2001

Creating ODBC Data Sources At Runtime

One of the drivers that comes with Clarion is the ODBC driver. Although Clarion deals with most of the problems of translating your file access code (e.g. OPEN, CLOSE, NEXT) into calls to the particular ODBC driver that looks after your data file, there is one area where Clarion ignores a potentially useful set of features of the ODBC design. These are the administration functions, which are required before you can access any data source through ODBC.

Posted Wednesday, May 09, 2001

Reader Comments Now Available On All Articles

You can now add your own comments to all ClarionMag and COL Archive articles.

Posted Tuesday, May 08, 2001

Weekly PDF for April 30 - May 6, 2001

All Clarion Magazine articles for April 30 - May 6, 2001.

Posted Monday, May 07, 2001

Replicating IDLE: All Quiet on the Keyboard?

Needing to have two inactivity timers running at the same time, Steve Parker goes in search of an IDLE equivalent.

Posted Thursday, May 03, 2001

The Clarion Advisor: API Tricks

Pierre Tremblay shows how to easily pass either a CSTRING or a NULL to the _strtok API parsing function.

Posted Thursday, May 03, 2001

Clarion Magazine's Publication Schedule

Clarion Magazine is still a weekly magazine, published 48 times per year, but we now post articles and news items throughout the week, rather than just on one day. Weekly PDFs appear the Monday following, and the weekly summary notices are also emailed on Mondays (except when the Monday falls on a statutory holiday).

Posted Wednesday, May 02, 2001

Introduction to SQL - Part 4

In Part 4 of this series, Dave Harms answers some questions about the differences between developing for flat file and SQL databases.

Posted Tuesday, May 01, 2001

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

View Recently Posted Reader Comments

Published 2001-05-31

This page lists all recently-posted reader comments. The comments are listed in reverse chronological order (newest first), but only under each article. If you're looking back more than a few days the overall order of the messages may *appear* to be out of sequence.

Display all reader comments posted in the last

Article	View Recently Posted Reader Comments	Posted	Friday, June 01, 2001
		by	Ralph Johnston

Nice addition Dave!

Now we'll have "recent posts" junkies, just like newsgroups junkies and e-mail junkies! <g>

[Join the discussion](#)

Article	View Recently Posted Reader Comments	Posted	Friday, June 01, 2001
		by	Tom Hebenstreit

Checking out the grouping/sorting of messages.<g>

[Join the discussion](#)

Article	View Recently Posted Reader Comments	Posted	Friday, June 01, 2001
		by	Dave Harms

This is the first release of this feature. If you have any suggestions, post them here!

[Join the discussion](#)

Article	The Novice's Corner: Understanding	Posted	Monday, June 04, 2001
----------------	--	---------------	-----------------------

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

[EQUATES.CLW \(Part 2\)](#)

by

Carl Barnes

CW Assistant has an Equate Viewer/Helper that helps get a handle on the 1500+ equates in Clarion by organizing them into 50 categories. For larger groups like "Prop:" and "Event:" it puts them in sub-categories like: Window, Field, List, At. It is free to use in the unregistered version. For download or info www.carlbarnes.com/cwa.htm.

At tip from Jim DeFabia was that if you have custom colors you use frequently you can default their equates in EQUATES.CLW and then use them in the IDE. You can get about 400 standard HTML colors as Clarin Equates at <http://home.powertech.no/sylkie/downloads.htm>.

CWA also has a color designer that lets you see and play with colors and color equates. Also free to use in the unregistered version.

Dave, I hope plugging products is OK here. I got tired of digging around in Equates.CLW and Property.CLW and made a better way.

[Join the discussion](#)

Article [The Novice's Corner: Understanding EQUATES.CLW \(Part 2\)](#)

Posted Monday, June 04, 2001

by Carl Barnes

test, delete me

[Join the discussion](#)

Article [Loading DLLs At Runtime - Part 3](#)

Posted Thursday, May 31, 2001

by Bruce Johnson

Thanks for this series Larry! It couldn't have come at a better time! Just exactly what we needed this week! 1 gem like this makes my whole ClarionMag subscription worthwhile. And I get more than 1 in a year!!

Cheers
Bruce Johnson
CapeSoft

[Join the discussion](#)

Article [Reading Tables With ADO](#)

Posted Wednesday, May 30, 2001

by Dave Harms

An alternate fix to the one in the updated zip (which moved the OVERed group out of the queue) is to insert this code:

?list{ PROP:Format} = ALL('2L',2*16)

after opening the window.

[Join the discussion](#)

Article	Reading Tables With ADO	Posted	Wednesday, May 30, 2001
		by	Dave Harms

I've updated the source - there was a problem (at least under C5.5) with the OVERed group inside the queue), which caused a parameter typing error when the listbox tried to display the queue.

[Join the discussion](#)

Article	Quickbooks-Style Date Fields	Posted	Monday, June 04, 2001
		by	Carl Barnes

Template Writer Utility (TWriter.EXE) would be good for this type of example. It takes existing code, extracts the embed code and writes you a template frame work that is a good starting point. At least you have all of the #AT's in pretty good shape.

You will find TWriter.EXE in your BIN directory and the install should have created a shortcut to it under the Tools folder.

The Template Writer Utility was shipped in C55 with the help broken. You must START TW.HLP from within the C55Bin directory to view it.

In C5 this was know as the Template Wizatron (TW.EXE). It's not really a true Wizatron but does help use Wizatrons by helping you make templates which you can then use in Wizatrons.

[Join the discussion](#)

Article	Quickbooks-Style Date Fields	Posted	Monday, June 04, 2001
		by	Carl Barnes

Didn't you read Bruce's article on CASE? <g>
Below is the IF changed to a CASE statement.

If the date is zero I think Quicken pops up with today which is a nice feature. The Code below sets the date to TODAY if it is currently zero using CHOOSE(~ %ControlUse...

I would recode as:

```
IF ~ %Control{ PROP:ReadOnly}
  UPDATE !So we get what he typed before +/-
  CASE KeyCode()
  OF 443 OROF PlusKey
    %ControlUse=CHOOSE(~ %ControlUse,TODAY(),%ControlUse+ 1)
```

```

OF 189 OROF MinusKey
  %ControlUse=CHOOSE(~%ControlUse,TODAY(),%ControlUse-1)
END
DISPLAY(%Control)
END

```

original code

```

IF %ControlUse <> 0 |
  AND %Control{ PROP:ReadOnly} <> TRUE
  IF KeyCode() = 443 OR KeyCode() = PlusKey
    %ControlUse += 1
  ELSIF KeyCode() = 189 OR KeyCode() = MinusKey
    %ControlUse -= 1
  END
  DISPLAY(%Control)

```

If you were going to have a lot of dates another interesting way to do it would be with "?" code like below so each new date adds just 1 new OROF line of code. (And the rest of the code remains unchanged.)

```

IF EVENT()=EVENT:AlertKey
  CASE Field()
  OF ?My:Date1 << tpw gen
  OROF ?My:Date2 << tpw gen
  OROF ?My:Date3 << tpw gen
  IF ~?{ PROP:ReadOnly}
    UPDATE
    CASE Keycode()
    OF 443 OROF PlusKey
      CHANGE(?,
CHOOSE(~ CONTENTS(?),TODAY(),CONTENTS(?)+1))
    OF 189 OROF MinusKey
      CHANGE(?, CHOOSE(~ CONTENTS(?),TODAY(),CONTENTS(?)-
1))
    OF Tkey
      CHANGE(?, TODAY())
    END
    DISPLAY(?)
  END
  END
  END
  END
END

```

[Join the discussion](#)

Article Quickbooks-Style Date Fields	Posted Tuesday, May 29, 2001
by	Andrew Guidroz II

To Mike ...

Exactly. I really wish I had thought longer about the article title as the neat part about the template isn't in its QuickBooks functionality. It is more how the little template tricks in it have tons of other applications.

[Join the discussion](#)

Article [Creating Elliptical Windows in Clarion](#)

Posted Wednesday, May 30, 2001

by James Cooke

It is really nice to see such a clear demonstration of good WINAPI implementation. This is a topic that has not been very prominent - up till now!
Thanks!

[Join the discussion](#)

Article [Four DLLs And An Executable](#)

Posted Friday, June 01, 2001

by Dave Harms

That sounds vaguely like a null reference problem. Where did this olex.dll come from?

[Join the discussion](#)

Reader Comments

[Add a comment](#)

**This is the first release of this feature. If you have any...
Checking out the grouping/sorting of messages.
Nice addition Dave! Now we'll have "recent posts"...**

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free

CLARION
*online*published by
CoveComm Inc.

Clarion MAGAZINE

The Clarion Advisor: SUB Tricks

by **Dave Harms and John Morter**

Published 2001-05-31

Have you noticed the negative position option for the second parameter of the SUB function? You can use this option to check the contents of the end of a string. SUB has the following syntax:

```
SUB(string,position,length)
```

The first parameter is the string you're extracting a substring from; the second parameter is the starting position of that substring; and the third parameter is the number of characters to return, beginning with the starting position.

If you use a negative number for the position, SUB uses the absolute value of that number relative to the end of the string, and then steps *backwards* for the number of characters specified by the length parameter. If, for instance, you want check for a trailing backslash on a directory name, you can use the following code to extract the last character from the variable:

```
IF SUB(PathName,-1,1) = ' \'
```

The above code will only work correctly if you use a CSTRING, however. If you use a STRING, you'll need to clip the trailing spaces:

```
IF SUB(CLIP(PathName),-1,1) = ' \'
```

Do you have a programming tip of interest to Clarion Magazine readers? Send it to advisor@clarionmag.com.

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

John Morter is a member of the Victorian Clarion Users Group (Melbourne, Australia). His moneymaking day job doesn't actually involve Clarion (at least not

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

officially), but Clarion occupies a lot of his spare time as a hobby to keep his techo-developer background up to date. He sails in the bay during the summer on his racing catamaran named Flat Chat, which is Australian slang for "at top speed" - or "at high velocity".

Reader Comments

[Add a comment](#)

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

The Novice's Corner: Understanding EQUATES.CLW (Part 2)

by Dave Harms

Published 2001-05-31

In any programming environment it's useful to represent certain commonly-used values as constants. In Clarion, the `EQUATE` keyword defines such a constant, and the two places you'll find most of these equates are in `EQUATES.CLW` (naturally) and `PROPERTY.CLW`, both of which are in the Clarion `libsrc\` directory. I've already covered the [first half](#) of `PROPERTY.CLW` – here's what's left.

Sound equates

In Windows, standard sounds correspond to sound settings as defined in the Control Panel. In a default Windows installation, where neither the user nor any of the user's software has mucked about with the sound settings, all of the `BEEP` equates will, most likely, call `chord.wav`, except for `BEEP:SystemDefault` which calls `ding.wav`.

```
BEEP:SystemDefault      EQUATE (0000H) ! ding.wav
BEEP:SystemHand         EQUATE (0010H) ! chord.wav
BEEP:SystemQuestion     EQUATE (0020H) ! chord.wav
BEEP:SystemExclamation  EQUATE (0030H) ! chord.wav
BEEP:SystemAsterisk     EQUATE (0040H) ! chord.wav
```

Spin box equates

The `REJECT` equates are used to test for invalid input on `SPIN` controls. To test for an invalid value use something like the following:

```
IF EVENT() = EVENT:Rejected
  CASE REJECTCODE()
    OF REJECT:RangeHigh
    ...

! Above top range on SPIN
REJECT:RangeHigh      EQUATE(1)
```

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

```

! below bottom range on SPIN
REJECT:RangeLow      EQUATE(2)
! Other range error
REJECT:Range         EQUATE(3)
! Invalid input
REJECT:Invalid       EQUATE(4)

```

Color equates

There are two kinds of color equates in EQUATES.CLW. The first set lists the sixteen basic Windows colors.

```

COLOR:Black          EQUATE (0000000H)
COLOR:Maroon         EQUATE (0000080H)
COLOR:Green          EQUATE (0008000H)
COLOR:Olive          EQUATE (0008080H)
COLOR:Navy           EQUATE (0800000H)
COLOR:Purple         EQUATE (0800080H)
COLOR:Teal           EQUATE (0808000H)
COLOR:Gray           EQUATE (0808080H)
COLOR:Silver         EQUATE (0C0C0C0H)
COLOR:Red             EQUATE (0000FFH)
COLOR:Lime           EQUATE (000FF00H)
COLOR:Yellow         EQUATE (000FFFFH)
COLOR:Blue           EQUATE (0FF0000H)
COLOR:Fuschia        EQUATE (0FF00FFH)
COLOR:Aqua           EQUATE (0FFFF00H)
COLOR:White          EQUATE (0FFFFFFFH)

```

More interesting to most developers than the basic colors, however, are the standard uses of colors. When you use one of the following equates as a color attribute, the actual color used will depend on the colors assigned to standard objects through the Windows control panel. I learned one particular good use for standard colors from Jeff Slarve. When Jeff has an entry field he wants to display as read-only, he sets the Skip and Read Only checkboxes, and sets the background color of the field to COLOR:BTNFACE, which is also the default color for the window background. This way the field is still visible (and can be copied), but clearly not modifiable.

```

COLOR:NONE           EQUATE (-1)
COLOR:SCROLLBAR      EQUATE (80000000H)
COLOR:BACKGROUND     EQUATE (80000001H)
COLOR:ACTIVECAPTION  EQUATE (80000002H)
COLOR:INACTIVECAPTION EQUATE (80000003H)
COLOR:MENU            EQUATE (80000004H)
COLOR:WINDOW         EQUATE (80000005H)
COLOR:WINDOWFRAME    EQUATE (80000006H)
COLOR:MENUTEXT       EQUATE (80000007H)
COLOR:WINDOWTEXT     EQUATE (80000008H)
COLOR:CAPTIONTEXT    EQUATE (80000009H)
COLOR:ACTIVEBORDER   EQUATE (8000000AH)
COLOR:INACTIVEBORDER EQUATE (8000000BH)
COLOR:APPWORKSPACE   EQUATE (8000000CH)

```

COLOR:HIGHLIGHT	EQUATE (8000000DH)
COLOR:HIGHLIGHTTEXT	EQUATE (8000000EH)
COLOR:BTNFACE	EQUATE (8000000FH)
COLOR:BTNSHADOW	EQUATE (80000010H)
COLOR:GRAYTEXT	EQUATE (80000011H)
COLOR:BTNTEXT	EQUATE (80000012H)
COLOR:INACTIVECAPTIONTEXT	EQUATE (80000013H)
COLOR:BTNHIGHLIGHT	EQUATE (80000014H)

Runtime control creation equates

Because Clarion provides an easy-to-use window formatter, few of us need to create controls at runtime. But if you have a large number of controls to display, or the number or type of controls on the window is variable, you may want to create these controls on the fly. This subject is worthy of an article all on its own, but for now I'll just refer you to the Help for CREATE(return new control created).

CREATE:sstring	EQUATE (1)
CREATE:string	EQUATE (2)
CREATE:image	EQUATE (3)
CREATE:region	EQUATE (4)
CREATE:line	EQUATE (5)
CREATE:box	EQUATE (6)
CREATE:ellipse	EQUATE (7)
CREATE:entry	EQUATE (8)
CREATE:button	EQUATE (9)
CREATE:prompt	EQUATE (10)
CREATE:option	EQUATE (11)
CREATE:check	EQUATE (12)
CREATE:group	EQUATE (13)
CREATE:list	EQUATE (14)
CREATE:combo	EQUATE (15)
CREATE:spin	EQUATE (16)
CREATE:text	EQUATE (17)
CREATE:custom	EQUATE (18)
CREATE:menu	EQUATE (19)
CREATE:item	EQUATE (20)
CREATE:radio	EQUATE (21)
CREATE:menubar	EQUATE (22)
CREATE:application	EQUATE (24)
CREATE>window	EQUATE (25)
CREATE:report	EQUATE (26)
CREATE:header	EQUATE (27)
CREATE:footer	EQUATE (28)
CREATE:break	EQUATE (29)
CREATE:form	EQUATE (30)
CREATE:detail	EQUATE (31)
CREATE:ole	EQUATE (32)
CREATE:droplist	EQUATE (33)
CREATE:dropcombo	EQUATE (34)
CREATE:progress	EQUATE (35)
CREATE:sheet	EQUATE (37)
CREATE:tab	EQUATE (38)

```

CREATE:panel           EQUATE (39)
CREATE:sublist        EQUATE (CREATE:list + 0100H)
CREATE:toolbar        EQUATE (128)

```

Font and charset equates

The font and charset equates are meant to be used with the `FONT` attribute, which can be added to any window or control (not just `TOOLBARS`, as the Help suggests). The `FONT` equates listed below are font style equates for the stroke (thin, regular, bold), fixed width, and other style attributes. `FONT:weight` is, as far as I know, only used in the Web templates to test for the font weight (boldness of stroke).

```

FONT:thin             EQUATE (100)
FONT:regular          EQUATE (400)
FONT:bold             EQUATE (700)
FONT:weight           EQUATE (07FFH)
FONT:fixed            EQUATE (0800H)
FONT:italic           EQUATE (01000H)
FONT:underline        EQUATE (02000H)
FONT:strikeout        EQUATE (04000H)

```

The following `FONT` equates represent new functionality as of Clarion 5.5a, and are passed to the last parameter of `FontDialog` and `FontDialogA` to restrict the set of fonts the dialog should present for selection.

```

FONT:Screen           EQUATE(0)
FONT:Printer          EQUATE(1)
FONT:Both             EQUATE(2)
FONT:TrueTypeOnly     EQUATE(4)
FONT:FixedPitchOnly   EQUATE(8)

```

The `CHARSET` equates specify Windows standard character sets, and are a parameter of the `FONT` attribute:

```

CHARSET:ANSI          EQUATE ( 0)
CHARSET:DEFAULT       EQUATE ( 1)
CHARSET:SYMBOL        EQUATE ( 2)
CHARSET:MAC           EQUATE ( 77)
CHARSET:SHIFTJIS      EQUATE (128)
CHARSET:HANGEUL       EQUATE (129)
CHARSET:JOHAB         EQUATE (130)
CHARSET:GB2312        EQUATE (134)
CHARSET:CHINESEBIG5   EQUATE (136)
CHARSET:GREEK         EQUATE (161)
CHARSET:TURKISH       EQUATE (162)
CHARSET:HEBREW        EQUATE (177)
CHARSET:ARABIC        EQUATE (178)
CHARSET:BALTIC        EQUATE (186)
CHARSET:CYRILLIC      EQUATE (204)
CHARSET:THAI          EQUATE (222)
CHARSET:EASTEUROPE    EQUATE (238)

```

CHARSET: OEM EQUATE (255)

Drawing equates

You can draw rudimentary graphics with Clarion, although for serious work you'll probably want to go with API calls, or better yet a third party product (like CapeSoft's Draw) that makes drawing Windows graphics easy. For simple lines, Clarion does let you specify the standard line styles, using PEN equates. Look at the SETPENSTYLE function, as well as SETPENCOLOR and SETPENWIDTH.

PEN:solid	EQUATE (0)
PEN:dash	EQUATE (1)
PEN:dot	EQUATE (2)
PEN:dashdot	EQUATE (3)
PEN:dashdotdot	EQUATE (4)
PEN:null	EQUATE (5)
PEN:insideframe	EQUATE (6)

Logic equates

Not much to say here, but if you've ever wondered how Clarion understands the keywords TRUE and FALSE, now you know.

FALSE	EQUATE (0)
TRUE	EQUATE (1)

List zone equates

I really should have included these equates in Part 1, as they're control-specific. If you need detailed information about mouse movements over a list box, you'll typically add an ALRT attribute to the list box (such as ALRT(MouseLeft)) and then test for a variety of list properties the list box receives EVENT:AlertKey. Several of these properties are PROPLIST:MouseUpRow, PROPLIST:MouseDownRow, PROPLIST:MouseUpField, and PROPLIST:MouseDownField. These properties tell you which row and column the mouse was on when the alerted key was pressed or released. But there are other areas on the list box such as headers and possibly tree list icons. To trap mouse movements relative to these areas you use PROPLIST:MouseDownZone, PROPLIST:MouseMoveZone, and PROPLIST:MouseUpZone, comparing the values against the equates listed below.

LISTZONE:field	EQUATE(0)
LISTZONE:right	EQUATE(1)
LISTZONE:header	EQUATE(2)
LISTZONE:expandbox	EQUATE(3)
LISTZONE:tree	EQUATE(4)
LISTZONE:icon	EQUATE(5)
LISTZONE:nowhere	EQUATE(6)

Button equates

Again, I should probably have included the button equates in Part 1 with the icon equates, since you'll usually use these together with icon equates in a MESSAGE() statement. Button equates not only tell the MESSAGE() function which button(s) to display, but optionally which button will be the default. For more information, see the Help for MESSAGE(return message box response).

```
BUTTON:OK                EQUATE (01H)
BUTTON:YES                EQUATE (02H)
BUTTON:NO                 EQUATE (04H)
BUTTON:ABORT              EQUATE (08H)
BUTTON:RETRY              EQUATE (10H)
BUTTON:IGNORE             EQUATE (20H)
BUTTON:CANCEL             EQUATE (40H)
BUTTON:HELP               EQUATE (80H)
```

Data type equates

There Windows data types signed and unsigned have different values in 16 bit and 32 bit applications. EQUATES.CLW uses the OMIT and COMPILE directives to create the correct equates depending on which kind of application you're compiling. Also included is an equate for the BOOL data type.

```
    OMIT( '***' , _WIDTH32_)
SIGNED                EQUATE (SHORT)
UNSIGNED              EQUATE (USHORT)
_nopos                EQUATE (08000H)
    ***
    COMPILE( '***' , _WIDTH32_)
SIGNED                EQUATE (LONG)
UNSIGNED              EQUATE (LONG)
_nopos                EQUATE (080000000H)
    ***
BOOL                  EQUATE (SIGNED)
```

Directory equates

When you're working with file directories, you'll most likely use the DIRECTORY function, which returns a queue containing a file directory listing. This is an overloaded function which can take either of two queue definitions, one for 8.3 filenames, and the other for long filenames. These are typed queue definitions, so you can't use them directly. Instead you'll create a queue derived from the type, and pass that to the DIRECTORY function.

The ff_ equates let you mask the returned list of files so that you only see files of the type you specified. You can add these equates together to get the desired combination of attributes.

```
ff_:NORMAL            EQUATE (0)
ff_:READONLY          EQUATE (1)
ff_:HIDDEN            EQUATE (2)
```

```

ff_:SYSTEM                EQUATE(4)
ff_:DIRECTORY             EQUATE(10H)
ff_:ARCHIVE               EQUATE(20H)
ff_:LFN                   EQUATE(80H)

!Old 8.3 filename support

ff_:queue    QUEUE,PRE(ff_),TYPE
name         string(13)
date        long
time        long
size        long
attrib      byte
                END

!full filename support

FILE:MaxFileName EQUATE(256)
FILE:MaxFilePath EQUATE(260)

FILE:Queue    QUEUE,PRE(FILE),TYPE
Name          STRING(FILE:MaxFileName)
ShortName     STRING(13)
Date          LONG
Time          LONG
Size          LONG
Attrib        BYTE
                END

```

File dialog equates

One of the real headaches in past editions of Clarion was getting a file dialog to return just a directory, instead of a file in the directory. Happily, there is now `FILE:Directory` equate which you can pass to `FileDialog`. If you don't pass a value at all (on the fourth, or flag, parameter), `FileDialog` shows an Open dialog. Other dialog flag options available include: `FILE:Save` - save a file; `FILE:KeepDir` - save and restore the current directory (important if you're opening and closing files and assuming that the files are in the current directory); `FILE:NoError` - do not report an error on a save if overwriting a file, on an open if the file doesn't exist; `FILE:Multi` - allow selection of multiple files delimited by spaces if using short filenames, or vertical bars if long filenames; and, of course, the above-mentioned `FILE:Directory`.

```

FILE:Save        EQUATE(1)
FILE:KeepDir     EQUATE(2)
FILE:NoError     EQUATE(4)
FILE:Multi       EQUATE(8)
FILE:LongName    EQUATE(10H)
FILE:Directory   EQUATE(20H)

```

The OLE queue

Another typed queue is `oleQ`, which you pass to the `OLEDIRECTORY`

function to get a list of all installed OLE servers or OCX controls.

```
oleQ          QUEUE,TYPE
name          CSTRING(64)
clsid        CSTRING(64)
progid       CSTRING(64)
END
```

The third and optional parameter to OLEDIRECTORY takes the following equates which lets you choose whether to include 16 bit controls, 32 bit controls, or both.

```
OCX:default   EQUATE(0)
OCX:16bit     EQUATE(1)
OCX:32bit     EQUATE(2)
OCX:1632bit   EQUATE(3)
```

Match equates

The Clarion MATCH function is a relatively new addition to the language that lets you compare two strings using a variety of techniques, each of which has a Match equate. A MATCH:Simple does per-character comparison, case sensitive unless you use MATCH:Simple + MATCH:NoCase. MATCH:Wild evaluates the first string against the second string which can contain * and ? characters. MATCH:Regular evaluates regular expression operators in the second string, against the first string. Regular expressions are an enormously powerful technique for matching text patterns. MATCH:Soundex will do a "sounds like" match using a standard soundex algorithm.

```
Match:Simple   EQUATE(0)
Match:Wild     EQUATE(1)
Match:Regular  EQUATE(2)
Match:Soundex  EQUATE(3)
Match:NoCase   EQUATE(10H)
```

Paper equates

The following equates are passed to PRINTER{PROPRINT:PAPER} to specify the paper size the report will use.

```
! Letter 8 1/2 x 11 in
PAPER:LETTER          EQUATE(1)
! Letter Small 8 1/2 x 11 in
PAPER:LETTERSMALL    EQUATE(2)
! Tabloid 11 x 17 in
PAPER:TABLOID        EQUATE(3)
! Ledger 17 x 11 in
PAPER:LEDGER          EQUATE(4)
! Legal 8 1/2 x 14 in
PAPER:LEGAL           EQUATE(5)
! Statement 5 1/2 x 8 1/2 in
PAPER:STATEMENT       EQUATE(6)
```

```

! Executive 7 1/4 x 10 1/2 in
PAPER:EXECUTIVE           EQUATE(7)
! A3 297 x 420 mm
PAPER:A3                   EQUATE(8)
! A4 210 x 297 mm
PAPER:A4                   EQUATE(9)
! A4 Small 210 x 297 mm
PAPER:A4SMALL             EQUATE(10)
! A5 148 x 210 mm
PAPER:A5                   EQUATE(11)
! B4 250 x 354
PAPER:B4                   EQUATE(12)
! B5 182 x 257 mm
PAPER:B5                   EQUATE(13)
! Folio 8 1/2 x 13 in
PAPER:FOLIO               EQUATE(14)
! Quarto 215 x 275 mm
PAPER:QUARTO              EQUATE(15)
! 10x14 in
PAPER:10X14                EQUATE(16)
! 11x17 in
PAPER:11X17                EQUATE(17)
! Note 8 1/2 x 11 in
PAPER:NOTE                 EQUATE(18)
! Envelope #9 3 7/8 x 8 7/8
PAPER:ENV_9                EQUATE(19)
! Envelope #10 4 1/8 x 9 1/2
PAPER:ENV_10               EQUATE(20)
! Envelope #11 4 1/2 x 10 3/8
PAPER:ENV_11               EQUATE(21)
! Envelope #12 4 \276 x 11
PAPER:ENV_12               EQUATE(22)
! Envelope #14 5 x 11 1/2
PAPER:ENV_14               EQUATE(23)
! C size sheet
PAPER:CSHEET               EQUATE(24)
! D size sheet
PAPER:DSHEET               EQUATE(25)
! E size sheet
PAPER:ESHEET               EQUATE(26)
! Envelope DL 110 x 220mm
PAPER:ENV_DL               EQUATE(27)
! Envelope C5 162 x 229 mm
PAPER:ENV_C5               EQUATE(28)
! Envelope C3 324 x 458 mm
PAPER:ENV_C3               EQUATE(29)
! Envelope C4 229 x 324 mm
PAPER:ENV_C4               EQUATE(30)
! Envelope C6 114 x 162 mm
PAPER:ENV_C6               EQUATE(31)
! Envelope C65 114 x 229 mm
PAPER:ENV_C65              EQUATE(32)
! Envelope B4 250 x 353 mm
PAPER:ENV_B4               EQUATE(33)
! Envelope B5 176 x 250 mm

```

```

PAPER:ENV_B5                EQUATE(34)
! Envelope B6  176 x 125 mm
PAPER:ENV_B6                EQUATE(35)
! Envelope 110 x 230 mm
PAPER:ENV_ITALY            EQUATE(36)
! Envelope Monarch 3.875 x 7.5 in
PAPER:ENV_MONARCH         EQUATE(37)
! 6 3/4 Envelope 3 5/8 x 6 1/2 in
PAPER:ENV_PERSONAL        EQUATE(38)
! US Std Fanfold 14 7/8 x 11 in
PAPER:FANFOLD_US          EQUATE(39)
! German Std Fanfold 8 1/2 x 12 in
PAPER:FANFOLD_STD_GERMAN  EQUATE(40)
! German Legal Fanfold 8 1/2 x 13 in
PAPER:FANFOLD_LGL_GERMAN  EQUATE(41)
PAPER:LAST                 EQUATE(41)
PAPER:USER                 EQUATE(256)

```

A TPS equate

Here's a lonely little TPS file equate. `TPSREADONLY` sets the TPS file to read-only when using the ODBC driver. See the TopSpeed Database Driver documentation for usage.

```
TPSREADONLY                EQUATE(1)
```

Driver option equates

The Driver option equates let you query a file at runtime to determine which features the file driver supports, using `file{PROP:SupportsOp,option}` where *option* is one of the following. Note that this group uses an `ITEMIZE` structure to automatically assign equate numbers, beginning with 1. Periodically the numbering is restarted. The second set of equates lets you determine which data types a given driver supports.

```

ITEMIZE(1),PRE(DriverOp)
ADD                EQUATE
BOF                EQUATE
BUILDfile          EQUATE
APPEND             EQUATE
BUILDdyn           EQUATE
BUILDkey           EQUATE
CLOSE              EQUATE
COMMIT             EQUATE
COPY               EQUATE
CREATE             EQUATE
DELETE            EQUATE
DUPLICATE          EQUATE
EMPTY              EQUATE
EOF                EQUATE
GETfilekey         EQUATE
GETfileptr         EQUATE
GETkeypтр         EQUATE

```

HOLD	EQUATE
LOCK	EQUATE (20)
LOGOUT	EQUATE (22)
NAME	EQUATE
NEXT	EQUATE
OPEN	EQUATE
PACK	EQUATE
POINTERfile	EQUATE
POINTERkey	EQUATE
FLUSH	EQUATE
PUT	EQUATE
PREVIOUS	EQUATE
RECORDSfile	EQUATE
RECORDSkey	EQUATE
BUILDdynfilter	EQUATE
RELEASE	EQUATE (36)
REMOVE	EQUATE
RENAME	EQUATE
ROLLBACK	EQUATE (40)
SETfile	EQUATE
SETfilekey	EQUATE
SETfileptr	EQUATE
SETkey	EQUATE
SETkeykey	EQUATE
SETkeyptr	EQUATE
SETkeykeyptr	EQUATE
SHARE	EQUATE
SKIP	EQUATE
UNLOCK	EQUATE
ADDlen	EQUATE
BYTES	EQUATE
GETfileptrlen	EQUATE
PUTfileptr	EQUATE
PUTfileptrlen	EQUATE
STREAM	EQUATE
DUPLICATEkey	EQUATE
WATCH	EQUATE
APPENDlen	EQUATE
SEND	EQUATE
POSITIONfile	EQUATE
POSITIONkey	EQUATE
RESETfile	EQUATE
RESETkey	EQUATE
NOMEMO	EQUATE
REGETfile	EQUATE
REGETkey	EQUATE
NULL	EQUATE
SETNULL	EQUATE
SETNONNULL	EQUATE
SETproperty	EQUATE
GETproperty	EQUATE
GETblobdata	EQUATE (75)
PUTblobdata	EQUATE
BLOBSIZE	EQUATE
SETblobproperty	EQUATE

```

GETblobproperty    EQUATE
BUFFER             EQUATE
SETviewfields     EQUATE
CLEARfile         EQUATE
RESETviewfile     EQUATE
BUILDevent        EQUATE
SETkeyproperty    EQUATE
GETkeyproperty    EQUATE
DOproperty        EQUATE( 88 )
DOkeyproperty     EQUATE
DOblobproperty    EQUATE
VIEWSTART         EQUATE( 92 )
VIEWSTOP          EQUATE
GETNULLS          EQUATE( 96 )
SETNULLS          EQUATE
GETSTATE          EQUATE
RESTORESTATE      EQUATE
CALLBACK          EQUATE
FREESTATE         EQUATE( 102 )
DESTROY           EQUATE( 104 )
    END

```

```

! Data Type Equates for use with
! file{PROP:SupportsType, DataType:n}

```

```

    ITEMIZE(1),PRE(DataType)
BYTE              EQUATE
SHORT             EQUATE
USHORT           EQUATE
DATE             EQUATE
TIME             EQUATE
LONG             EQUATE
ULONG           EQUATE
SREAL           EQUATE
REAL            EQUATE
DECIMAL         EQUATE
PDECIMAL        EQUATE
BFLOAT4         EQUATE( 13 )
BFLOAT8         EQUATE
STRING          EQUATE( 18 )
CSTRING         EQUATE
PSTRING         EQUATE
MEMO            EQUATE
BLOB            EQUATE( 27 )
    END

```

Other Includes

The EQUATES.CLW file isn't the only place to find equates – many are specific to the property syntax (PROPERTY.CLW), and there's another set just for reports (PRNPROP.CLW). But those are subjects for another time...

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

Reader Comments

[Add a comment](#)

test, delete me

CW Assistant has an Equate Viewer/Helper that helps get a...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free

CLARION
*online*published by
CoveComm Inc.

Clarion MAGAZINE

Loading DLLs At Runtime - Part 3**by Larry Sand**

Published 2001-05-29

[Last week](#) I explained the workings of LoadLibClass, a Clarion class I've created to make it easy to call DLL functions by address. This week I'll put this class to work.

A common request is to find the free disk space for a drive. Windows 95 provides a function, GetDiskFreeSpace, which does this. However, at the time it was first released, Windows 95 only supported partitions smaller than 2GB.

MSDN warns that "the GetDiskFreeSpace function may return misleading values." (on partitions greater than 2 GB). It also warns that "Even on volumes that are smaller than 2 gigabytes, the values stored into **lpSectorsPerCluster*, **lpNumberOfFreeClusters*, and **lpTotalNumberOfClusters* values may be incorrect." Furthermore, it goes on to say that the GetDiskFreeSpace function is superseded by GetDiskFreeSpaceEx but that function is only available on Windows 95 OSR2, 98, ME, Windows NT 4.0 and higher, including Windows 2000.

So, what's a weary programmer who wants to find the free disk space, regardless of Windows version, to do? Use that shiny new load library class of course!

Implementing the load library class

I designed DiskInfoClass, presented in this section, to illustrate how to implement the load library class. It is the beginning of a class that you can extend with your own methods to gather disk information. First I'll provide a description of the relevant Windows API functions, and then I'll explain how to wrap these in class methods to make calling them as simple as calling any other Clarion method.

Two functions, different methods, same result

In the Win32 API, functions with names that contain the "Ex" suffix are extended versions of older functions and they supercede the

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

non-"Ex" version. You prototype the GetDiskFreeSpaceEx function like this:

```
GetDiskFreeSpaceEx(LONG pszDirectoryName=0, |
    *ULARGE_INTEGER lpFreeBytesAvailable, |
    *ULARGE_INTEGER lpTotalNumberOfBytes, |
    *ULARGE_INTEGER lpTotalNumberOfFreeBytes |
    ), BOOL, RAW, PASCAL, DLL(_fp_)
```

The first parameter is a pointer to a `cstring` that contains the directory name. This can be any directory on the disk, a UNC path name, or a null pointer for the current directory's disk. Prototype this parameter as a long to make it easy to pass a null pointer. In normal use, assign the `ADDRESS()` of the string or a zero to the `pszDirectoryName` parameter before you call the function.

The next three parameters are pointers to `ULARGE_INTEGER`s. This variant of the disk free space function returns the disk information as 64-bit integers, sometimes referred to as `QUAD`s. Most languages do not natively support these integers, including Clarion. One way to use them is to prototype them as a group of two unsigned longs (the `RAW` attribute on the function prototype instructs Clarion to only pass the address), like this:

```
ULARGE_INTEGER GROUP, TYPE
LowDw           ULONG
HighDw          ULONG
                END
```

Notice that the declaration places the high double word after the low double word in the group. Intel's architecture stores the least significant byte first. This byte order is known as [Little Endian](#) or Intel order. I'll explain how to convert these integers into something useful in Clarion later in this article.

`GetDiskFreeSpace` is the older variant of the two Windows API disk space functions. It is prototyped like this:

```
GetDiskFreeSpace(LONG pszDirectoryName=0, |
    *ULONG lpSectorsPerCluster, |
    *ULONG lpBytesPerSector, |
    *ULONG lpNumberOfFreeClusters, |
    *ULONG lpTotalNumberOfClusters |
    ), BOOL, RAW, PASCAL, DLL(_fp_)
```

Like `GetDiskFreeSpaceEx`, this function takes a pointer to a `cstring` that contains the root directory as its first parameter. MSDN states that "a drive specification such as "C:" cannot have a trailing backslash." However, in testing "C:\\" is accepted. Even more perplexing is that use of the drive specification without the trailing backslash produces the error code 123, for an invalid directory name, on some versions of Windows. The function also accepts a null pointer to refer to the disk drive of the current directory. Furthermore, this function doesn't support UNC path

names on Windows 95 before OSR2.

Unlike `GetDiskFreeSpaceEx`, `GetDiskFreeSpace` accepts pointers to four unsigned LONGs to return its disk information instead of the 64-bit integers. Some simple math on these parameters yields the disk space information.

Notice that both of these prototypes include the `DLL(_fp_)` attribute. Remember that this means that you must declare a function pointer variable for each. As I showed in [Part 1](#), you must declare these variables with the `NAME` attribute specifying the label of the appropriate function in single quotes. Therefore, define the function pointer variables like this:

```
fpGetDiskFreeSpaceEx
LONG,AUTO,NAME('GetDiskFreeSpaceEx')
fpGetDiskFreeSpace LONG,AUTO,NAME('GetDiskFreeSpace')
```

With that background information in your pocket, you're ready to examine how the `DiskInfoClass` wraps the two disk free space functions into a single method call. First, examine how the `Init` method initializes the objects.

Listing 5. The Init method

```
DiskInfoClass.Init    PROCEDURE()
RetVal    LONG,AUTO
    CODE
    RetVal = 1
    fpGetDiskFreeSpaceEx = 0
    fpGetDiskFreeSpace    = 0
    SELF.Kernel &= NEW LoadLibClass
    IF NOT SELF.Kernel &= NULL
        RetVal = |
            SELF.Kernel.LlcLoadLibrary('kernel32.dll', |
                Method:GetModuleHandle)
        IF SELF.Kernel.LibraryLoaded()
            fpGetDiskFreeSpaceEx = |
                SELF.Kernel.LlcGetProcAddress( |
                    'GetDiskFreeSpaceExA')
            fpGetDiskFreeSpace =
                SELF.Kernel.LlcGetProcAddress( |
                    'GetDiskFreeSpaceA')
        END
    END
    RETURN RetVal
```

Notice that the class definition (see `DiskInfo.inc`) contains the `Kernel` property, a reference to the load library class `&LoadLibClass`. I like to name the reference variable after the library it manages, `kernel32.dll` in this case. The method creates an instance of the `LoadLibClass` and assigns it to the `Kernel` reference property.

After the method instantiates the `LoadLibClass`, the `Init` method tests that the reference to the `Kernel` object is not null before it attempts to reference the `Kernel` object. Whenever the reference is null, the object instantiation failed, and any use of that reference will cause a GPF. In addition, the method returns non-zero for failure.

When the `DiskInfo` object has a valid reference to a `Kernel` object, the `Init` method initializes the `Kernel` object by calling the `LlcLoadLibrary` method with the DLL file name and the load method. In this case, the method uses the `Method:GetModuleHandle` load method to get the handle of the module. After the method successfully loads `kernel32.dll`, it assigns the addresses of `GetDiskFreeSpaceEx` and `GetDiskFreeSpace` to their respective function pointer variables. It's not an error if `fpGetDiskFreeSpaceEx` is null, it just means that the `GetDiskFreeSpaceEx` function is not available on this system.

After the `Init` method completes successfully, the `DiskInfo` object contains a reference to an initialized `Kernel` object (an instance of the `LoadLibClass` managing the handle to the Windows kernel). Furthermore, the function pointer variables contain the address of their respective disk space function.

After your code calls the `DiskInfoClass` `Init` method, you must call the `GetDiskSpace` method, passing a `DISK_SPACE` group with the `szDirectoryName` element set to the directory of interest.

Listing 6. The `GetDiskSpace` method

```
DiskInfoClass.GetDiskSpace |
  PROCEDURE(*DISK_SPACE DiskSpace)
RetVal          LONG,AUTO
  CODE
? ASSERT(NOT SELF.Kernel &= NULL)
  IF SELF.Kernel.LibraryLoaded()
    RetVal = SELF.DiGetDiskFreeSpaceEx(DiskSpace)
  IF RetVal
    RetVal = SELF.DiGetDiskFreeSpace(DiskSpace)
  END
ELSE
  RetVal = 1
END
RETURN RetVal
```

The public `GetDiskSpace` method (see Listing 6) attempts to call the `DiGetDiskFreeSpaceEx` method as its first choice. If that method fails, it assumes that `GetDiskFreeSpaceEx` is not available on this OS and attempts to call the `DiGetDiskFreeSpace` method. When successful, the `GetDiskSpace` method fills the three disk space elements of the `DISK_SPACE` group with the results from one of the two API functions and returns zero. If for some reason initialization of the `Kernel` object failed and you call this method, it

returns a non-zero result.

The `DiGetDiskFreeSpaceEx` method wraps the Windows API function `GetDiskFreeSpaceEx`. Consider this code for the `DiGetDiskFreeSpaceEx` method: .

Listing 7. The `DiGetDiskFreeSpaceEx` method

```

DiskInfoClass.DiGetDiskFreeSpaceEx  |
    PROCEDURE(*DISK_SPACE DiskSpace)
i64FreeBytesAvailable    LIKE(ULARGE_INTEGER)
i64TotalBytes            LIKE(ULARGE_INTEGER)
i64TotalFreeBytes       LIKE(ULARGE_INTEGER)
RetVal                  LONG,AUTO
pszDirectoryName        LONG,AUTO
    CODE
    RetVal = 0
    DiskSpace.FreeBytesAvailable = 0
    DiskSpace.TotalBytes = 0
    DiskSpace.TotalFreeBytes = 0
    IF fpGetDiskFreeSpaceEx
        pszDirectoryName = |
            CHOOSE(DiskSpace.szDirectoryName<>' ',|
                ADDRESS(DiskSpace.szDirectoryName), 0)
        IF GetDiskFreeSpaceEx(pszDirectoryName,      |
            i64FreeBytesAvailable, |
            i64TotalBytes,          |
            i64TotalFreeBytes)
            !Convert the 64 bit unsigned integers
            ! into decimals
            ULIntToDec(DiskSpace.FreeBytesAvailable, |
                i64FreeBytesAvailable)
            ULIntToDec(DiskSpace.TotalBytes, |
                i64TotalBytes)
            ULIntToDec(DiskSpace.TotalFreeBytes, |
                i64TotalFreeBytes)
        ELSE
            RetVal = SELF.Kernel.GetLastError()
        END
    ELSE
        RetVal = 1
    END
    RETURN RetVal

```

One of the first things that the `DiGetDiskFreeSpaceEx` method does is to test if the function pointer is not null. When it is null, it usually means that the `GetDiskFreeSpaceEx` function is not available so the object should use the `DiGetDiskFreeSpace` method instead.

Before the method can call the `GetDiskFreeSpaceEx` function, it must assign the address of `pDiskInfo.szDirectoryName` to `pszDirectoryName`. Because the function was prototyped to take a long integer for this parameter, you must use the `ADDRESS`

function, like this:

```
pszDirectoryName = |
    CHOOSE(DiskSpace.szDirectoryName<>' ', |
    ADDRESS(DiskSpace.szDirectoryName), 0)
```

The CHOOSE function returns the address of the string if it is not blank, or a zero when it is blank. The zero instructs GetDiskFreeSpaceEx to return the disk information of the current directory.

As I mentioned earlier, GetDiskFreeSpaceEx requires pointers to three 64-bit unsigned integers as parameters, so the method declares these as ULARGE_INTEGERS. GetDiskFreeSpaceEx passes these 64-bit unsigned integers by address and fills them with the disk space information for the specified directory.

Before you can use these 64-bit unsigned integers in your program, you must convert them into a format that Clarion understands. One way to do this is to use a variable declared as DECIMAL(21). This decimal variable is sufficient to hold the maximum value (18,446,744,073,709,551,615) of a 64-bit unsigned integer. To convert 64-bit unsigned integers into decimals, you must shift the high double word 32 bits to the left and add the lower 32 bits to that result.

Normally when you want to shift a value a certain number of bits, you use the BSHIFT function. In this case, however, BSHIFT will not work. So how will you convert the group into the decimal? (hint... shifting the bits left 32 times is the same as multiplying by 2^{32} .) Consider the listing for the ULIntToDec helper procedure:

```
ULIntToDec PROCEDURE(*DECIMAL dResult, |
                    *ULARGE_INTEGER I64)
TWO_TO_32 EQUATE(4294967296) !2^32
CODE
    dResult = I64.HighDw * TWO_TO_32 + I64.LowDw
RETURN
```

This procedure takes the high order 32 bits, multiplies it by the constant TWO_TO_32, and then adds the lower 32-bit value. Assume that you use this on a large drive and it returns the following values:

```
I64.HighDw = 00000008H
I64.LowDw = 8C8A6000H
```

I'm going to use hexadecimal notation to make it easier to see what is happening in this example.

I64.HighDw starts out as a 32-bit value:

```
00000008H
```

After it is multiplied by 2^{32} the result is:

```
00000008 00000000H
```

Why did that work? Don't worry, there won't be any assembly language this time. Clarion's automatic type conversion cast the unsigned long to a decimal before it performed the multiplication. Because of this type cast, no data are lost.

When you add the two values together, you get the desired result:

```
  00000008 00000000H
+ 00000000 8C8A6000H
-----
  00000008 8C8A6000H
```

or in decimal:

```
  34,359,738,368
+  2,357,878,784
-----
 36,717,617,152
```

That is a whopping 34.1 GB of disk space.

If the `DiGetDiskFreeSpaceEx` method fails, the class falls back on the `DiGetDiskFreeSpace` method. Whenever a program made with this class runs on Windows 95 before OSR2, `DiGetDiskFreeSpace` always executes. If you compare the code in Listing 8 to the code Listing 7, you will see that they have more in common than not.

Listing 8. The `DiGetDiskFreeSpace` method

```
DiskInfoClass.DiGetDiskFreeSpace |
    PROCEDURE(*DISK_SPACE DiskSpace)
SectorsPerCluster      ULONG,AUTO
BytesPerSector         ULONG,AUTO
FreeClusters          ULONG,AUTO
Clusters              ULONG,AUTO
RetVal                LONG,AUTO
pszDirectoryName      LONG,AUTO
CODE
RetVal = 0
DiskSpace.FreeBytesAvailable = 0
DiskSpace.TotalBytes = 0
DiskSpace.TotalFreeBytes = 0
IF fpGetDiskFreeSpace
    pszDirectoryName = |
        CHOOSE(DiskSpace.szDirectoryName<>'', |
        ADDRESS(DiskSpace.szDirectoryName), 0)
    IF GetDiskFreeSpace(pszDirectoryName, |
        SectorsPerCluster, |
        BytesPerSector, |
        FreeClusters, |
        Clusters)
        DiskSpace.TotalFreeBytes = |
```

```

        BytesPerSector * |
        SectorsPerCluster * |
        FreeClusters
    DiskSpace.TotalBytes = |
        BytesPerSector * |
        SectorsPerCluster * |
        Clusters
    DiskSpace.FreeBytesAvailable = |
        DiskSpace.TotalFreeBytes
ELSE
    RetVal = SELF.Kernel.GetLastError()
END
END
RETURN RetVal

```

The only significant differences are in `GetDiskFreeSpace`'s parameters. As noted before, `GetDiskFreeSpace` accepts four unsigned longs that represent the disk geometry. To calculate values similar to `GetDiskFreeSpaceEx`, you only need to find the product of `BytesPerSector` and `SectorsPerCluster`, and then multiply it by the `FreeClusters` or `Clusters` for the `TotalFreeBytes` and `TotalBytes` respectively. Since `GetDiskFreeSpace` does not return information about available free bytes, the method sets `FreeBytesAvailable` equal to `TotalFreeBytes`.

Finally, the `Destruct` method releases the memory allocated for the Kernel object. Disposing of the Kernel object fires its `Destruct` method, which calls its `LlcFreeLibrary` method.

In Summary

You now know the difference between run-time and load-time dynamic linking. You should also have a good understanding of when run-time linking is useful, what function pointers are, and how to create them to implement run-time dynamic linking in Clarion.

The load library base class presented in this article gives you a platform to derive and use via composition to give your class access to the Windows API or other external libraries.

You've seen how to implement the `LoadLibClass` in another class. The `DiskInfoClass` is the beginning of a class that you can extend to retrieve disk information. The `DiskInfoClass` illustrates how to dynamically load and call the Windows API `GetDiskFreeSpaceEx` function. Furthermore, it shows how to fall back on the `GetDiskFreeSpace` function if `GetDiskFreeSpaceEx` does not exist on the OS.

[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](#)

Thanks for this series Larry! It couldn't have come at a...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

Clarion News

File Explorer 1.7 Shipping

CapeSoft's File Explorer version 1.7 is now shipping. File Explorer is a way of including HTML in your program, either as a read-only browser or as an HTML editor. The main new feature is support for the recently released Adobe Acrobat 5 reader. If you use File Explorer in your app, then the client can have either Acrobat 4 or Acrobat 5 loaded and your program will work smoothly. FE auto-detects which OCX is loaded on the machine and behaves accordingly. For the duration of the Big Birthday Bash File Explorer costs \$89. This price expires on 31 May 2001. After that it's back to the normal price of \$99.

Posted Friday, May 25, 2001

NetTalk 1.0 Beta 12 With DUN Available

NetTalk 1.0 Beta 12 is now available, with Dial-Up Networking (DUN) support. Your program can automatically connect to another machine, making full use of the Windows built-in DUN features. It's even possible for your program to detect when a Dial-Up connection is made, and automatically piggy-back its own information across the link. The features supported are numerous. Gold release is expected late in June.

Posted Friday, May 25, 2001

CapeSoft Draw Version 1.0 Beta 2 Available

CapeSoft Draw is now in beta 2. There have been some questions about how Draw and Insight Graphing fit together. For the Insight users, rest assured. Insight isn't getting dropped in any

way. Insight will be changed over to use the new Draw engine in the near future. Of course drawing is a big part of drawing graphs, but Insight does much more than just draw. Its main achievement is in the way it collects the data from your data files, collates it, and then presents it. Although Insight will use the Draw engine internally, you won't need to purchase Draw in order to use Insight. Draw is however a major step forward for Insight as it allows for vertical text, and saving to PNG's (a web format). And of course Draw is very useful on its own. Draw creates graphics quickly, and the pictures don't have to be redrawn every time the window is moved or refreshed. Animation is flicker-free, and you can mix BMP files with drawing functions. CapeSoft Draw is priced at \$99, however for the duration of the CapeSoft Birthday Bash you can get it for \$59. From 1 June onwards, until the end of the beta program, the price will be \$79.

Posted Friday, May 25, 2001

[CapeSoft Mailer 1.0 Beta 1 Released](#)

CapeSoft Mailer is a bulk mail management system. It's not a SPAM tool, and shouldn't be used as such. Rather it allows you to create and maintain various mailing lists. You can then send an email to the people on the list. Each person receives their own email, with no multiple recipients. Mailer allows you to create the Email in both HTML and Text formats. If you have recipients that prefer to receive text-only emails then Mailer automatically sends the text only version to them. If a person opts-out of your list then their address is not physically deleted, rather it is marked as "excluded". This prevents them being added to the list again in the future. Although Mailer itself is not designed as an accessory, but rather as a full working program, there are some features available to Clarion developers. CapeSoft is shipping the DCT as part of the basic package. This means you're easily able to interface your program to the data files, building your own lists etc. Secondly source code is available (for \$499). Mailer makes extensive use of NetTalk, File Explorer, File Manager 2, SecWin, and WinEvent. These accessories are not included in the mailer source code price. CapeSoft Mailer costs \$99 per site. Resellers received a 50% discount on the second and subsequent copies.

Posted Friday, May 25, 2001

[CapeSoft MessageBox Version 1.0 beta 1 Released](#)

Message Box is a simple tool which allows you to customize your

MESSAGE, STOP and HALT windows. It gives you more control over the look of the window, but at the same time it integrates into your existing programs in a transparent way. The main features of the CapeSoft MessageBox are: source only - no precompiled DLLs; compatible with C4, C5, C5.5, ABC and Legacy; compatible with other CapeSoft products, like Makeover, Ezhelp and Special Agent. MessageBox adds features to the Standard message box, like auto timeouts, sound, logging, etc. INcluded is a utility that lets you build the MessageBox and view it in real time. MessageBox is priced at \$49, but will be on special for \$39 until the end of the (probably very short) beta program.

Posted Friday, May 25, 2001

[The CapeSoft Big Birthday Bash Ends Soon](#)

On 1 May 2001 CapeSoft officially turned 10, and for the month of May all CapeSoft Clarion Accessories are on special.

Posted Friday, May 25, 2001

[One Week Left On FrameText Special](#)

FrameText from solid.software is available at a \$10 discount only until May 31 - after that date the price goes back to \$59.

Posted Thursday, May 24, 2001

[Search Engines Profile Exchange Updated](#)

Encourager Software has created an centralized information resource for software authors that need to submit their programs and web sites to the various search engines. Encourager Software grants a limited license for individuals or companies to install the Viewer Version of Product Scope 32 PRO on 5 computers or less and Search Engine Data Files without a paid registration. This is not a file submittal program. Rather, it is a collection of information designed to help you get to the sites in an organized fashion.

Posted Tuesday, May 22, 2001

[Buggy 2.1.4 Available](#)

An update to the Buggy bug tracking tool is available to all registered users. The trial version has also been updated.

Posted Wednesday, May 16, 2001

IMPEX 5.0 Adds HTML Export

IMPEX allows easy, user-controlled, drag-and-drop import and export of data using templates which automatically read file format information (such as Field Names, Field Types etc) from the Data Dictionary (export) or from the file header (import). File structures are constructed automatically. The demo will import any dBase or ASCII file so you can test it on your own data files. IMPEX will now output HTML as well, with end user control over page and table attributes, and inclusion of other files (i.e. header and footer). Other new features include: export to flat ASCII files; set field order for all exports; template locates field names; export to tab, pipe, comma and semicolon delimited ASCII. Demo available.

Posted Monday, May 14, 2001

FrameText Special Offer

FrameText, from solid.software, is now available at a \$10 discount through the end of May, 2001. FrameText is a library/template add-on that allows Clarion applications to display text on the client area of the MDI frame window (i.e. for displaying licensing information, advertisements, tips of the day, etc.). FrameText lets you specify: font name, style, charset and color; justification (left, center, right); shadow/light color for 3D effect (optional). Supports Clarion5 and 5.5, ABC and legacy templates, 16 and 32 bit target platforms, in DLL and LIB versions supporting both standalone and local compiles. Updates and support are free. Until Thursday, May 31, FrameText will be priced at \$49 instead of \$59. Also available at ClarionShop - <http://www.clarionshop.com>.

Posted Monday, May 14, 2001

Stealth Software Mail & Fax Upgrade Pricing Changes

From June 1st, Stealth Software we will be only selling the upgrade to the C55 version of the Mail and Fax templates through clarionshop.com (as well as new licences of course). Through clarionshop.com the upgrade will cost US\$25. If you purchase directly from Stealth before June 1st, you can still buy it for US\$20. So if you haven't yet upgraded from an earlier version to the Clarion 5.5 version of the templates, please either email

(cliff@vine.co.za) before the end May and get them for US\$20, or after May from clarionshop.com for US\$25.

Posted Thursday, May 10, 2001

Clarion To Excel Example

Anton Novikov has released a small class and example app showing how to transfer data from Clarion to Excel. YOU can write to one cell, one row, or an entire table. If you have problems with the download email Anton at anfront@chat.ru.

Posted Wednesday, May 09, 2001

CapeSoft Draw Version 1.0 Beta 1 Released

CapeSoft Draw Version 1.0 beta 1 released. This drawing engine is at the heart of the Insight graphing product. Some of the advantages of CapeSoft Draw are: speed; images to not have to be drawn every time the window is moved or refreshed; flicker-free animation; mixing BMP files with drawing functions; save results as BMP or PNG. CapeSoft Draw is priced at \$99, but during the CapeSoft Birthday Bash you can get it for \$59.

Posted Wednesday, May 09, 2001

The CapeSoft Big Birthday Bash

On 1 May 2001 CapeSoft officially turned 10. That's 10 whole years of Clarion programming going all the way back to Clarion Professional for Dos 2.0, including Clarion 3 for Dos, and the whole Windows line from CW 1.0. To celebrate this occasion CapeSoft is running the Big Birthday Bash all the way during the Month of May, with all CapeSoft Clarion Accessories at discounted prices.

Posted Wednesday, May 09, 2001

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the expresswritten consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca.

Clarion MAGAZINE

Creating Elliptical Windows in Clarion

by **Brice Schagane**

Published 2001-05-25

While attending ETC 2000, I heard James Fortune give an excellent talk on [User Assistance 2000](#). In his presentation, James made several recommendations regarding help and how it is best provided. I liked what I heard. James recommended CapeSoft's EzHelp, for its ability in providing contextual help to the end-user. As a result, I ran right out and purchased a copy of EzHelp, along with several other CapeSoft products.

James' presentation also included a discussion on installation programs. He recommended that any media, which contained multiple installation files should provide a single interface program, such as a CD browser. A CD browser is a small "teaser" application, which launches upon disc detection, and presents installation information to the end-user in a straight-forward manner. It can be used to launch other applications, such as setup programs, or provide additional product and service information.

When I installed CapeSoft's software, I immediately recognized their browser program and was especially impressed with the program's appearance. Instead of a standard, boring, square (or rectangular) window, they used an elliptical window. I was inspired!

Getting started

Now that I had the inspiration, I needed to figure out how to create elliptical windows within my own applications. I got out my handy-dandy WIN32 Programming API Bible (by Richard Simon, Waite Group Press) and went to work. After a little digging, I found an API function named `CreateEllipticRgn`. This sounded promising. The `CreateEllipticRgn` requires four parameters, one for each corner of a given rectangle, and returns a handle to the newly created region. It seemed only logical that I pass it the dimensions of a predefined window. By creating an elliptical region that is the same width and height as my window, I can easily approximate the boundaries of the resulting elliptic region. When populating controls to my window, I must position them such that, when I convert the window to an elliptic region, the controls are

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

still visible. In order for this to work, I would need to determine the coordinates and/or dimensions of the targeted window.

The next API function I stumbled upon is named `GetWindowRect`. That's pretty clear to me. That's exactly what I need, the rectangular coordinates of the window. The `GetWindowRect` function requires two parameters. The first parameter is the handle to a window. The second is a `RECT` structure and is used by the function to return the coordinates of the window's bounding rectangle. This function also returns a Boolean `TRUE`, if the function is successful, and `FALSE` if the function ultimately fails.

Trial and error

Now that I had the functions `GetWindowRect` and `CreateEllipticRgn`, I needed to put them to use. After a little tinkering around, I discovered that I had missed something. I was using the API calls to create an elliptical region, but the resulting region was not being applied to my window. I somehow needed to instruct Windows to restrict the boundaries of my window to that of a specific region. After searching vigorously, I found a function named `SetWindowRgn` in the MSDN Library (why this function isn't listed in the API Bible, I may never know).

The `SetWindowRgn` function sets the visible region of a window and specifies whether or not the window should be redrawn. This function requires three parameters. The first two parameters consist of the handle to a window, and the handle to a region, respectively. The window that is passed here, will be bound to the boundaries of the specified region. The third parameter is a Boolean flag (true or false), indicating whether or not the window should be redrawn. `SetWindowRgn` also returns an integer value indicating success (nonzero) or failure (zero).

Putting it to use

Surprisingly, the code required to create elliptical windows isn't complex at all. Now that I had established the foundations, I needed to find some practical use for an elliptical window. Writing a CD browser was somewhat pointless given the fact that I didn't have a need for one.

There are some differences between normal and elliptical windows, and that affected my decision on what kind of window to create. Unlike traditional windows, elliptical windows are best presented without a title bar. That means the user loses the ability to drag the window from one position to another and to close the window using a system menu on the title bar. While moving the window may or may not be an issue, I should always provide a mechanism for closing the window. I thought about using a traditional close button, but that might take away from the elliptical window's flare.

My next thought was an elliptical Splash window. Splash windows can be closed automatically after a specified amount of time. They

can also be setup to close at any time by simply clicking on it. An elliptical splash window sounded like it would make an ideal test subject.

For testing purposes, I used a splash window within the School Manager application, one of Clarion's sample applications.

Global Definitions

The first thing I needed to do was make sure that the application compiled to 32-bit, since the API calls are all 32-bit. Under Project|Properties, I verified that the target OS was set to Windows 32-bit.

Next, I defined the prototypes and equates for the API calls, using SoftVelocity's Windows API INCLUDE File Constructor (also known as the WinAPI program, located in the Clarion example files). I created an include file named 'WinRgnApi.clw', as shown in Listing 1, and saved it to Clarion's LibSrc directory.

Listing 1. Contents of WinRgnApi.clw Include file

```
SECTION('Equate')
HANDLE      EQUATE(UNSIGNED)
HRGN        EQUATE(HANDLE)
HWND        EQUATE(HANDLE)
BOOL        EQUATE(SIGNED)
HGDIOBJ     EQUATE(HANDLE)
RECT        GROUP,TYPE
left        SIGNED
top         SIGNED
right       SIGNED
bottom      SIGNED
            END
SECTION('Prototypes')
MODULE('Windows.DLL')
  GetWindowRect(HWND, *RECT),BOOL,PASCAL,RAW
  CreateEllipticRgn(SIGNED,SIGNED,SIGNED,SIGNED)|
    ,HRGN,PASCAL
  SetWindowRgn(HWND,HRGN,BOOL),SIGNED,PASCAL
END
```

Now that I had my Include file, I had to include it within my application. Under Global Properties|Embeds, I performed the following steps:

1. In the Section After Global INCLUDEs, I embedded the following source code:

```
INCLUDE('WinRgnApi.clw','Equate')
```

2. In the Section 'Inside the Global Map', I embedded the following source code:

```
INCLUDE('WinRgnApi.clw','Prototypes')
```

Up to this point, I had defined the elements necessary to call the API functions from within my application. Next, I had to write the code that actually defines an elliptical window, using the 'SplashIt' procedure's window as a building block.

Local Definitions

The process of converting a procedural window to an elliptical window required that I define some local variables to the procedure. Using either the Local Data dialog or the 'Data for the procedure' embed point, I defined the following five variables:

```
!Coordinates of the window's bounding rectangle
WinRect      LIKE(RECT)
DBSRgn       HRGN          !Handle to the New Region
DBSWidth     LONG          !Width of the window
DBSHeight    LONG          !Height of the window
SuccessFlag  BYTE          !Status of calls
```

Finally, I wrote the code necessary to create an elliptical window. In the first available embed point, after the window is opened (ThisWindow.Init at Priority 8030), I embedded this code:

```
SuccessFlag = GetWindowRect( |
    WindowName{PROP:HANDLE},WinRECT)
IF SuccessFlag
    DBSWidth = WinRect.right - WinRect.left
    DBSHeight = WinRect.bottom - WinRect.top
    DBSRgn = CreateEllipticRgn(0,0,DBSWidth,DBSHeight)
    SetWindowRgn(WindowName{PROP:HANDLE},DBSRgn,TRUE)
END!_IF
```

The first line of code retrieves the coordinates of the named window and returns TRUE if successful. I replaced WindowName with the actual label of my window. In this case, it is simply "window". The second line of code determines whether or not I should continue creating my elliptical window. If GetWindowRect is unsuccessful, the normal rectangular splash screen will display. However, due to the planned changes to the window's appearance, if GetWindowRect fails, the resulting window wouldn't be very pretty.

The two lines of code following IF SuccessFlag calculate the width and height of the original window by subtracting the left coordinate from the right coordinate and the top coordinate from the bottom coordinate. The fifth line of code is used to create an elliptic region within the current window. The first two zeros specify the x and y coordinates for the new region and each is relative to the top-left corner of the current window. As I mentioned previously, specifying the width and height to be the same as the current window will make it easier to position controls within the elliptical region. This, however, is not required.

The sixth line of code tells Windows to restrict the visible portion of the window to the boundaries of the region. The Boolean TRUE indicates that the window should be redrawn.

Let'er RIP!

Now I was ready to compile and run my application. The compiler generated a warning message: "Warning: calling function as procedure". This is because the `SetWindowRgn` function returns an integer value indicating success (nonzero) or failure (zero). To get rid of this message, you could either trap for an error by setting a variable equal to the return value, or add the `,PROC` attribute to the function's prototype. Neither change is required. The worst that could happen is that your window would remain square. To this day, I haven't encountered such a problem.

Once the application ran, I got an elliptical splash window, but it still needed a little work. By removing the two larger panels, adjusting the position of the other controls, and adding some color, I generated a more appealing window similar to that shown in Figure 1.



Figure 1. Elliptical Splash Window

Summary

In a few easy steps, I took a basic splash window and created a much more appealing elliptical window. You too, can quickly and easily, create elliptical windows within your own applications. Elliptical windows add a special little touch that can help improve your software's overall image. The possibilities are endless.

You may also be interested in taking a look at the `CreateRoundedRectRgn` API function. This function will allow you to create a rectangular region with rounded corners. If you're really artistic or just looking for a good challenge, try the `CreatePolygonRgn` API function, which can give your window the appearance of practically any shape you desire.

[Download the source code](#)

Brice Schagane works for the Kentucky Transportation Cabinet. He also runs a small computer company by the name of Ghost Solutions, Inc. Brice has been using Clarion since 1997.

Reader Comments

[Add a comment](#)

It is really nice to see such a clear demonstration of good...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Clarion MAGAZINE

Loading DLLs At Runtime - Part 2

by **Larry Sand**

Published 2001-05-22

[Last week](#) I explained the theory behind calling DLL procedures by address. This week I'll take you through the steps of actually calling such procedures.

There are five steps necessary to call a procedure by address after it is prototyped with a function pointer.

1. Load the DLL with LoadLibrary
2. Get the address of the procedure with GetProcAddress
3. Assign the address from step 2 to the function pointer variable
4. Call the procedure (The class does not perform this step)
5. Unload the DLL with FreeLibrary when done.

Listing 2 defines the load library class which manages the process of loading the DLL, getting the addresses of the procedures in the DLL, and finally unloading the DLL. The class doesn't provide a method to call procedures from the DLL. To do this, you must derive the class or use the class via composition.

Listing 2. Load library class definition (see LoadLib.inc)

```
LoadLibClass          CLASS,TYPE,MODULE('LoadLib.clw'),|
                      LINK('LoadLib.clw',|
                      _ABCLinkMode_),|
                      DLL(_ABCDllMode_)
LastError             ULONG,AUTO
LlcLoadLibrary        PROCEDURE(|
                      STRING ModuleFileName,|
                      UNSIGNED |
                      LoadMethod=Method:LoadLibrary|
                      ),LONG
LlcGetProcAddress    PROCEDURE(STRING sProcedureName)|
                      ,LONG
LlcFreeLibrary        PROCEDURE(),LONG,PROC
LibraryLoaded        PROCEDURE(),BOOL
GetLastAPIError      PROCEDURE(),ULONG
LoadMethod           UNSIGNED,PROTECTED
```

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

```

hModule                UNSIGNED(0),PROTECTED
szModuleFileName       &CSTRING,PROTECTED
szProcedureName        CSTRING(64),AUTO,PROTECTED
OnLoadLibraryFailure   PROCEDURE(),VIRTUAL,PROTECTED
OnGetProcAddressFailure PROCEDURE(),VIRTUAL,PROTECTED
Destruct               PROCEDURE(),PROTECTED
                        END

```

Note: You must declare the API prototypes and function pointers at Module or Global scope. Failure to do this will cause unpredictable behavior and your program will most likely cause a GPF. To do this, include the class header file (LoadLib.inc) in the module where you wish to call the methods. For an application, select the Module tab and then embed the include statement in the "Module Data Section" embed. Alternatively, if you need global access to the class, embed the include statement in the Global embeds "After Global INCLUDES" embed point.

Before the object can get the address of a procedure, it must first load the DLL or get the handle to a previously loaded DLL. Use the Windows API functions `LoadLibrary` or `GetModuleHandle` to load or get the handle to the DLL. Prototype these functions like this:

```

LoadLibrary(*CSTRING pszModuleFileName) |
    ,UNSIGNED,PASCAL,RAW,NAME('LoadLibraryA')
GetModuleHandle(*CSTRING pszModuleName) |
    ,UNSIGNED,PASCAL,RAW,NAME('GetModuleHandleA')

```

`LoadLibrary` maps the DLL named in the `pszModuleFileName` parameter into the program's address space, increments the DLL's usage count, and returns a handle to the DLL. Whenever a program loads a DLL into memory with `LoadLibrary`, Windows increments its usage count. It's this usage count that Windows uses to determine when to unload the DLL. Conversely, Windows decrements the usage count when you call `FreeLibrary` and it's only after the usage count reaches zero that Windows finally unloads the DLL from memory. The returned handle is simply a 32-bit integer that uniquely identifies the instance of the DLL.

In contrast, `GetModuleHandle` only returns the handle of a previously loaded module (DLL). It does not map the DLL into the address space of your program or increment its usage count. Whenever you use `GetModuleHandle` to get the handle to the DLL, you should not call `FreeLibrary`. The class manages this distinction for you. If another process unloads the DLL, the handle returned by `GetModuleHandle` can become invalid, so, use this option with caution. It's intended for modules that you *know* will not be unloaded by another process, such as the Windows kernel (kernel32.dll).

Like many Windows API functions, `LoadLibrary` and

`GetModuleHandle` come in two flavors, ANSI and Unicode. That's the reason their prototypes contain the `NAME` attribute, and specify `LoadLibraryA` and `GetModuleHandleA` as their respective external names. These are the ANSI versions of the functions; the Unicode versions are `LoadLibraryW` and `GetModuleHandleW`.

How will you know when there are two versions of an API function? You can always read the SDK header files (.h) for the function. If C header files make you see double, you can look at the end of the function's documentation on [MSDN](#). You'll find this line (or one like it) for functions that are present in ANSI and Unicode:

Unicode: Implemented as Unicode and ANSI versions on Windows NT/2000.

Furthermore, if the MSDN help for a function says "**Unicode:** Unicode only", or something similar, then you'll need to convert your ANSI strings to Unicode before calling that function, and back to Multi-Byte ANSI strings when done. See Jim Kane's [article](#) for more information.

This class does not follow the convention of using an `Init` method for initialization, nor does it use a `Kill` method to deallocate its resources. Instead, it performs its own initialization, and then cleans up after itself. (Now if I could just convince my children to do the same!) You perform the following steps to load a DLL, call a procedure, and then unload the DLL: First, call the `LlcLoadLibrary` method with the file name of the DLL. You only call this method once per library. Next, call the `LlcGetProcAddress` method with the name of each procedure in the managed DLL and assign it to the procedures function pointer variable. Finally, call the `LlcFreeLibrary` method or let the `Destruct` method complete this step for you.

Now I'll describe the implementation of each of these methods. The `LlcLoadLibrary` method is responsible for initializing the object and calling `LoadLibrary` or `GetModuleHandle` based on the optional `LoadMethod` parameter. If you don't specify the method, `LoadLibrary` is the default.

Listing 2. LlcLoadLibrary Method

```
LoadLibClass.LlcLoadLibrary PROCEDURE( |
    STRING sModuleFileName, |
    UNSIGNED |
    LoadMethod=Method:LoadLibrary)
RetCode      LONG, AUTO
CODE
RetCode = 1
IF NOT SELF.LibraryLoaded()
    SELF.szModuleFileName &= NEW CSTRING( |
        LEN(CLIP(sModuleFileName))+1)
    IF NOT SELF.szModuleFileName &= NULL
        SELF.szModuleFileName = CLIP(sModuleFileName)
```

```

        SELF.LoadMethod = |
            CHOOSE(LoadMethod < Method:Last AND |
                LoadMethod > 0, |
                LoadMethod, |
                Method:LoadLibrary)
    EXECUTE SELF.LoadMethod
    SELF.hModule = |
        LoadLibrary(SELF.szModuleFileName)
    SELF.hModule = |
        GetModuleHandle(SELF.szModuleFileName)
    END
END
IF SELF.LibraryLoaded()
    RetCode = 0
ELSE
    SELF.OnLoadLibraryFailure()
END
END
RETURN RetCode

```

The first thing that `LlcLoadLibrary` does is to check if it's already managing a DLL. If it is, `LibraryLoaded` returns true and the method doesn't do anything other than return 1 to indicate failure.

Note: This class follows the convention that a function returns zero (0) for success and Non-zero implies some kind of failure. Extended error information is stored in the classes' `LastError` property. These are the error codes returned by the Windows API function `GetLastError`. You can find the definition of these error codes on MSDN (** dh add link).

If the object isn't already managing a library, the method allocates memory for the module file name. The file name is passed as a string value parameter and `LoadLibrary` and `GetModuleHandle` require a `*cstring` for this parameter. This allows you to call the method with a constant string like this:

```
Di.LlcLoadLibrary('MyDLL.DLL')
```

Next, the method checks the validity of the load method value and assigns it to the `LoadMethod` property. The method defaults to `Method:LoadLibrary`, which is the safest method. The `EXECUTE` structure assigns the handle returned by `LoadLibrary` or `GetModuleHandle` to the `hModule` property. The class requires this value as a parameter of the API calls to get a procedure's address and to unload the library.

The object calls the virtual method `OnLoadLibraryFailure` whenever it cannot obtain the handle to the DLL. Derive the virtual `OnLoadLibraryFailure` method to handle this error as you see fit.

Before you can use a procedure from the DLL, you must assign the

procedure's address to the function pointer variable. To get the address use the `LlcGetProcAddress` method. The `LlcGetProcAddress` method wraps the Windows API function `GetProcAddress`; assigning the passed procedure name to a cstring, similar to `LlcLoadLibrary`, and testing the validity of the returned address. You prototype `GetProcAddress` like this:

```
GetProcAddress(UNSIGNED hModule, |
               *CSTRING pszProcName |
               ),LONG, PASCAL, RAW
```

The `LlcGetProcAddress` method takes two parameters, the handle to the module returned by `LoadLibrary` or `GetModuleHandle`, and a pointer to the procedure name. The second parameter can optionally accept the ordinal of the exported procedure. I leave it an exercise for you to overload this method to accept an ordinal instead of the procedures' name. Now consider how the class implements the `GetProcAddress` function.

Listing 3. LlcGetProcAddress method listing

```
LoadLibClass.LlcGetProcAddress PROCEDURE(|
                                STRING sProcedureName)
lpProcedure      LONG,AUTO
CODE
IF SELF.LibraryLoaded()
    SELF.szProcedureName = CLIP(sProcedureName)
    lpProcedure = GetProcAddress(SELF.hModule,|
                                SELF.szProcedureName)
    IF IsBadCodePtr(lpProcedure)
        lpProcedure = 0
        SELF.OnGetProcAddressFailure()
    END
ELSE
    lpProcedure = 0
END
RETURN lpProcedure
```

When called, the method ensures that the object is managing a DLL. That is, you already called the `LlcLoadLibrary` method. If the object is managing a DLL, the method assigns the procedure name passed as a string value parameter to the `szProcedureName` property (a cstring). The Windows API `GetProcAddress` function requires a cstring.

The method does not dynamically allocate the `szProcedureName` property as the `szModuleFileName` property is in the `LlcLoadLibrary` method. If you encounter procedure names greater than 63 characters, modify this property. The `LlcLoadLibrary` method set the `hModule` property to the handle of the DLL..

`GetProcAddress` returns an address or zero if it fails. Additionally, the method calls `IsBadCodePtr` to test for read access to returned

address. Since zero is an invalid address, `IsBadCodePtr` returns true. If your process does not have read access to the address, the method calls the `OnGetProcAddressFailure` method. Derive this method in your class to handle the error, if necessary. Some procedures are not present in all versions of Windows, in which case it is normal for `GetProcAddress` to fail. Your code can use this fact to call an alternate procedure. The section on implementing the `LoadLibClass` illustrates this technique.

Finally, the method returns the address of the procedure. Use this address to call the procedure from your program. When the return value is zero, the method failed to locate the address of the procedure.

If you wish to use the same instance of the object to manage another library, call `LlcFreeLibrary` before calling `LlcLoadLibrary`. When you are done with the object call `LlcFreeLibrary`, or let the `Destruct` method do it for you when the object is disposed. `LlcFreeLibrary` only frees the library once, so it doesn't matter if you explicitly call it as well as the `Destruct` method.

Listing 4. `LlcFreeLibrary` listing

```
LoadLibClass.LlcFreeLibrary PROCEDURE()
RetVal LONG,AUTO
CODE
DISPOSE(SELF.szModuleFileName)
RetVal = 0
IF SELF.LibraryLoaded()
    IF SELF.LoadMethod = Method:LoadLibrary
        IF NOT FreeLibrary(SELF.hModule)
            RetVal = 1
            SELF.LastError = GetLastError()
        END
    END
    SELF.hModule = 0
END
RETURN RetVal
```

`LlcFreeLibrary` disposes of the memory allocated for the module file name in `LlcLoadLibrary`. Then it does one of two things depending on the value of the `LoadMethod` property. The method calls the Windows API `FreeLibrary` when `LoadLibrary` returned the module handle. `FreeLibrary` decrements the DLL's usage count each time it's called. When the DLL's usage count reaches zero, Windows unloads the DLL from memory. On the other hand, the method simply sets the `hModule` property to zero when `GetModuleHandle` returns the module handle .

Now that you have this class to manage run-time dynamic linking, you will want to use it do something useful. Next week I'll show you how to call a function in a Windows DLL, by address.

[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](#)

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free

CLARION
*online*published by
CoveComm Inc.

Clarion MAGAZINE

Reading Tables With ADO

by **Dave Harms and Brian Staff**

Published 2001-05-21

Have you ever wanted to write a generalized utility to handle a data file which may exist on more than one backend database? Do you need a utility to handle a file when you don't have/or want a DCT layout? Have you ever wanted to use ADO (ActiveX Data Objects) in Clarion as a standard way of managing your data?

SoftVelocity is working on a set of templates that wrap up Andy Ireland's ADO code, and we're expecting great things. But you can already do ADO with surprisingly little effort, using the code Jim Kane described in his [COM articles](#) (all the code you need to make this ADO code work is, however, included in the downloadable source linked at the end of this article).

The RecordSet object

To use ADO, you create a `RecordSet` object which you use to retrieve data from one or more data files. A `RecordSet` object contains all the selected data as well as information about the data (metadata), such as field names, data types, and so on.

`RecordSets` are a bit like `ViewManager` objects, except that you have to tell a `ViewManager` explicitly which fields you're working with, whereas you can tell a `RecordSet` to get, say, all the fields in a file, and then you can ask the `RecordSet` what those fields are before you attempt to retrieve the data.

The `RecordSet`'s `Open` method takes five parameters, all of which are optional:

`Source` – a variant data type which can be one of many things: a SQL statement, a table name, a stored procedure, a URL, an ADO Command object, or a file or Stream object containing a persisted record set.

`ActiveConnection` – a variant data type holding a `Connection` object, or a connection string identifying the data source

`CursorType` – a cursor type enum (see below)

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

`LockType` – a lock type enum (see below)

`Options` – information on how the source should be handled (see below)

The sample application

The sample code (160 or so lines) we describe in this article will allow you to use an ADO `RecordSet` object to read a TopSpeed file - although this can be any data file if you have an ODBC driver for it - and place the data in a standard Clarion `LIST` control. You can download the code at the end of this article.

Here's how the application works. To begin with, you need to know some standard ADO equates.

The cursor location equates refer to the cursor, or marker, used to keep track of your current position in the record set. Client cursors seem to work much better than server cursors, especially in a multi-user environment.

```
!---- CursorLocationEnum Values ----
adUseServer          EQUATE(2)
adUseClient          EQUATE(3)
```

Besides cursor location, you also need to choose a cursor type. Cursors not only manage your current position, they may also need to manage changes to the database (deletes, inserts) and possibly communicate information about those changes to other users. You pass the cursor type to the `RecordSet` object's `Open` method.

The `adOpenDynamic` equate indicates a dynamic cursor. With this cursor you can navigate backwards and forwards through a `RecordSet`, and you can see any changes made by other users.

The `adOpenKeyset` equate indicates a cursor like a dynamic cursor, with the following differences: you can't access records others delete; you can't see records others add; you can see data others change.

The `adOpenStatic` equate indicates a static cursor. This cursor keeps its own copy of the records you manipulate, so you can't see any changes others make.

The default cursor type is `adOpenForwardOnly`, which is the same as a static cursor but only lets you scroll forward through the data set. Generally speaking, `adOpenForwardOnly` will result in faster operation than `adOpenStatic`, though at the obvious expense of functionality.

```
!---- CursorTypeEnum Values ----
adOpenForwardOnly    EQUATE(0)
```

```
adOpenKeyset           EQUATE(1)
adOpenDynamic          EQUATE(2)
adOpenStatic           EQUATE(3)
```

The `LockTypeEnum` values indicate how ADO handles concurrency, and you pass this value to the `RecordSet`'s `Open` method. If you use `adLockReadOnly`, you can't alter data at all; `adLockPessimistic` indicates that the data source will lock records as you retrieve the data to begin your edit; `adLockOptimistic` indicates that the data source will lock records only when you issue an update; and `adLockBatchOptimistic`, as you might guess, applies optimistic locking to batch updates.

```
!---- LockTypeEnum Values ----
adLockReadOnly         EQUATE(1)
adLockPessimistic      EQUATE(2)
adLockOptimistic       EQUATE(3)
adLockBatchOptimistic EQUATE(4)
```

The `CommandTypeEnum` values indicate the type of command used to query a database and return data in a `RecordSet` object. Like cursor type and lock type, the command type is passed to the `RecordSet`'s `Open` method.

The `adCmdText` equate indicates that the command passed to `Open` as the `Source` parameter is to be evaluated as a command. For instance, if you wish to execute a `SELECT * FROM MyTable`, you pass this string to `Open` as the `Source`, with an `Options` parameter of `adCmdText`. The `adCmdTable` equate indicates that the `Source` is a table name, and all fields in the table should be returned in the `RecordSet`. The `adCmdStoredProc` equate, as you'd expect, indicates that the `Source` is a stored procedure that should be executed. The default is `adCmdUnknown`.

```
!---- CommandTypeEnum Values ----
adCmdUnknown           EQUATE(0008h)
adCmdText              EQUATE(0001h)
adCmdTable             EQUATE(0002h)
adCmdStoredProc        EQUATE(0004h)
```

The sample application includes a few more equates and variables, most of which are straightforward. Note that the ADO `RecordSet` is called `oFileX`, and is declared as an OLE object using Jim Kane's `oleTclType` base class.

```
oFileX &oleTclType
```

The purpose of the sample application is to connect to a data source using ADO and retrieve the records into a queue for display in a window. The first step is to set the name of the database, and the value of the connect string. The sample application uses the developer version of the TPS ODBC driver, but you can change this to any data source you like. Note that although the TPS file is actually called `cust.tps`, it's only necessary to supply the table

name, not the physical file name (that is defined in the [ODBC data source definition](#)):

```
ThisFileName = 'cust'
strConnect = 'DRIVER={{ Topspeed Developer version}}
;DBQ=C:\data\'
```

Now it's time to create the ADO RecordSet object:

```
oFileX &= NEW oleTClType
oFileX.init('ADODB.RecordSet',0)
```

Next, create the SQL statement, and pass it to the RecordSet object with the appropriate equates:

```
sql = 'SELECT * FROM ' & ThisFileName
oFileX.CallMethod('Open("'"&sql&'", "'&strConnect&'",
"'&adOpenForwardOnly&'", "'&adLockReadOnly&'",
"'&adCmdText&'"'))
```

You don't need to know anything about the data source in advance – the application will examine the resulting data and retrieve the column (field) names and other data. This code returns the number of columns in the table:

```
Cols = oFileX.GetProp('Fields.Count')
```

The sample code loops through the available columns and retrieves the column names (up to the maximum supported by the display queue), using them to format the queue's columns:

```
LOOP j = 1 TO CLIP(Cols)
  s1 = |
    oFileX.GetProp('Fields('& j-1 &').Name')
  ?p1{PROP:Text}= CLIP(s1)
  ?List{PROPList:Format,j} = |
    (?p1{PROP:Width} + 4) |
    & 'L(2)M|~Hdr~(2)@s30@'
  ?List{PROPList:Header,j} = s1
  IF j >= max THEN BREAK.
END
```

The following loop retrieves the record data from RecordSet and adds it to the display queue:

```
LOOP
  MyEOF = oFileX.GetProp('EOF')
  IF MyEOF <> 0 THEN BREAK.
  LOOP j = 1 TO CLIP(Cols)
    qs.colx[j] = |
      oFileX.GetProp('Fields('& j-1 &').Value')
    IF j >= max THEN BREAK.
  END
  ADD(qs)
```

```
oFileX.CallMethod('MoveNext()')
END
```

Now just close, kill, and dispose of the RecordSet:

```
oFileX.CallMethod('Close()')
oFileX.Kill()
DISPOSE(oFileX)
```

That's how easy it is to use ADO with Clarion. The SoftVelocity ADO templates, soon to be released, will make it even easier. In the meantime, you may want to use some of this code for your own ADO explorations.

[Download the source](#)

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

[Brian Staff](#) was born and raised in Rugby, England, and lived for 28 years in Vancouver, Canada. He worked too many years for Honeywell on mainframes, and spent six years as an independent developer, including four years developing software for DirecTv. Brian currently develops point of sale systems and web applications for JDA Software in Phoenix, Arizona (it's a dry heat). A member of Team Topspeed since Feb 1996, Brian is also the author of the Xplore templates and is a coach and volunteer web site developer for the local soccer community. He is married to Valerie, has three soccer-playing daughters, and is a former international level rugby referee.

Reader Comments

[Add a comment](#)

Excellent Article!

Thanks! This is all Brian's code - I just wrote it up. I'm...

It might serve a better purpose if you wrapped this code in...

Ross - that's what Andy's supposed to be doing

Stop arguing already, I'm on the case!

<bg>

Ross, You're right of course. But Andy is working on...

I've updated the source - there was a problem (at least...

An alternate fix to the one in the updated zip (which moved...

www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

The Novice's Corner: Understanding EQUATES.CLW (Part 1)

by Dave Harms

Published 2001-05-18

In any programming environment it's useful to represent certain commonly-used values as constants. In Clarion, the `EQUATE` keyword defines such a constant, and the two places you'll find most of these equates are in `EQUATES.CLW` (naturally) and `PROPERTY.CLW`, both of which are in the Clarion `libsrc\` directory.

`PROPERTY.CLW` contains equates specifically used with the property syntax, while `EQUATES.CLW` contains many general-purpose equates. The following is a brief annotation of the contents of `EQUATES.CLW`. I've changed the order of some of the equates, as they're not always grouped by function in the file.

Event equates

Clarion equates typically have standardized prefixes, and event equates are no exception. Events are also sometimes called messages; for instance, when you click your mouse on the scroll bar of a browse window, the Windows operating system detects the click and sends an appropriate scroll event to your application. You can find out which event has occurred by calling the `EVENT()` function.

The following are field-level events; that is, they are specific to controls on a window, rather than the window itself.

`EVENT:Accepted` is probably the most commonly-used event, since it occurs whenever the user enters data or makes any other selection on a control, and then moves to another control. You'll typically use template embeds instead of testing for this event directly.

```
EVENT:Accepted      EQUATE (01H)
```

The following are browse control events, and for the most part Clarion developers don't deal with these directly. In fact, there are two distinct ways Clarion deals with browse events. In a list box

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

without the IMM attribute, Clarion handles all the browse events internally, and the programmer doesn't have any direct access to them. That's fine for small list boxes where all the data fits in the queue associated with the list box. But most browses are page loaded. In this case the IMM attribute must be on the browse box so that browse events make it through to the ACCEPT loop, where (typically) the ABC classes and template-generated code respond to the events, and display data accordingly.

```
EVENT:NewSelection    EQUATE (02H)
EVENT:ScrollUp        EQUATE (03H)
EVENT:ScrollDown      EQUATE (04H)
EVENT:PageUp          EQUATE (05H)
EVENT:PageDown        EQUATE (06H)
EVENT:ScrollTop        EQUATE (07H)
EVENT:ScrollBottom    EQUATE (08H)
EVENT:Locate           EQUATE (09H)
EVENT:ScrollDrag       EQUATE (14H)
EVENT:ScrollTrack      EQUATE (1DH)
EVENT:ColumnResize    EQUATE (1EH)
```

The mouse movement events are generated when you have a REGION control with the IMM attribute. If you're creating something like a drawing program, you'll need to use this technique to generate the mouse tracking events (and use the MOUSEX and MOUSEY functions to track the mouse's position).

```
EVENT:MouseDown       EQUATE (01H)
EVENT:MouseUp         EQUATE (0aH)
EVENT:MouseIn         EQUATE (0bH)
EVENT:MouseOut        EQUATE (0cH)
EVENT:MouseMove       EQUATE (0dH)
```

A VBX on a window needs a way to notify the window that it (the VBX) has some important information to convey. That's the purpose of EVENT:VBXevent. There are equates for various specific VBX events as well.

```
EVENT:VBXevent        EQUATE (0eH)
VBXEVENT:Click        EQUATE (0)
VBXEVENT:DblClick     EQUATE (1)
VBXEVENT:GotFocus     EQUATE (4)
VBXEVENT:KeyDown      EQUATE (5)
VBXEVENT:KeyPress     EQUATE (6)
VBXEVENT:KeyUp        EQUATE (7)
VBXEVENT:LostFocus    EQUATE (8)
VBXEVENT:MouseDown    EQUATE (9)
VBXEVENT:MouseMove    EQUATE (10)
VBXEVENT:MouseUp      EQUATE (11)
```

Alert keys are hot keys that are active for the entire window. Each alert key press causes two events to trigger, first EVENT:PreAlertKey, then EVENT:AlertKey. If you issue a CYCLE statement after EVENT:PreAlertKey, then EVENT:AlertKey will be

processed, otherwise it will be discarded. Among other things, you can use this feature to override standard Windows hot keys, but only under specific circumstances.

```
EVENT:AlertKey      EQUATE (0fH)
EVENT:PreAlertKey  EQUATE (10H)
```

Drag and drop processing is easy to implement in Clarion, and probably a bit underused. See Mike Hanson's [Drag & Drop article](#) in the [COL archives](#) for more.

```
EVENT:Dragging      EQUATE (11H)
EVENT:Drag          EQUATE (12H)
EVENT:Drop          EQUATE (13H)
```

On tree-format list controls, the following events indicate that the user is in the process of clicking, or already has clicked on an expand/contract icon in the list.

```
EVENT:Expanding     EQUATE (16H)
EVENT:Contracting   EQUATE (17H)
EVENT:Expanded      EQUATE (18H)
EVENT:Contracted    EQUATE (19H)
```

If the user has entered bad data (invalid data for a string picture, out of range spin value), the Clarion runtime will generate a Rejected event.

```
EVENT:Rejected      EQUATE (1AH)
```

The following events indicate that a list with a drop attribute is dropping down, or has dropped down.

```
EVENT:DroppingDown  EQUATE (1BH)
EVENT:DroppedDown   EQUATE (1CH)
```

The Selecting and Selected events indicate that a control is about to receive, or has already received, input focus.

```
EVENT:Selecting     EQUATE (1FH)
EVENT:Selected       EQUATE (101H)
```

When you move from one tab to another, the Clarion runtime generates a TabChanging event, which you can use to prepare the new tab or do other processing.

```
EVENT:TabChanging   EQUATE (15H)
```

The following events apply to the window as a whole, rather than to a specific field. Whenever you open or close a window, minimize, restore, or maximize, or switch to or from a window, that window is notified of the action with an appropriate message.

```
EVENT:CloseWindow   EQUATE (201H)
```

```

EVENT:CloseDown      EQUATE (202H)
EVENT:OpenWindow     EQUATE (203H)
EVENT:OpenFailed     EQUATE (204H)
EVENT:LoseFocus      EQUATE (205H)
EVENT:GainFocus      EQUATE (206H)
EVENT:Suspend        EQUATE (208H)
EVENT:Resume         EQUATE (209H)
EVENT:Move           EQUATE (220H)
EVENT:Size           EQUATE (221H)
EVENT:Restore        EQUATE (222H)
EVENT:Maximize       EQUATE (223H)
EVENT:Iconize        EQUATE (224H)
EVENT:Completed      EQUATE (225H)
EVENT:Moved          EQUATE (230H)
EVENT:Sized          EQUATE (231H)
EVENT:Restored       EQUATE (232H)
EVENT:Maximized      EQUATE (233H)
EVENT:Iconized       EQUATE (234H)

```

Time events are a way to cause code to execute at specific intervals. To create a timer event, just put the `TIMER(interval)` attribute on the window with a suitable interval value.

```
EVENT:Timer          EQUATE (20BH)
```

DDE stands for Dynamic Data Exchange, which is a Windows standard for inter-application communication. Not widely used by Clarion developers, DDE does have its uses. Among other things, you can use DDE to exercise some control over the Clarion environment (this is how batch compilers work).

```

EVENT:DDErequest     EQUATE (20CH)
EVENT:DDEadvise      EQUATE (20DH)
EVENT:DDEdata        EQUATE (20EH)
EVENT:DDEcommand     EQUATE (20FH)!same as DDEexecute
EVENT:DDEexecute     EQUATE (20FH)
EVENT:DDEpoke        EQUATE (210H)
EVENT:DDEclosed      EQUATE (211H)
!DDE link types
DDE:auto             EQUATE (0)
DDE>manual           EQUATE (-1)
DDE:remove           EQUATE (-2)

```

Dockable toolboxes can be created in Clarion – see the Clarion `examples\docktb` directory for an example, or read Steffen Rasmussen's article on Outlook-style menus (<http://www.clarionmag.com/cmag/v2/v2n8outlookmenu1.html>).

```

EVENT:Docked         EQUATE (235H)
EVENT:Undocked       EQUATE (236H)

```

Later in `EQUATES.CLW` you'll see other docking-related equates, not for events, but for properties to be used with `PROP:Dock`.

```
DOCK:Left            EQUATE(1)
```

```
DOCK:Top           EQUATE ( 2 )
DOCK:Right        EQUATE ( 4 )
DOCK:Bottom       EQUATE ( 8 )
DOCK:Float        EQUATE (16 )
DOCK:All          EQUATE (31 )
```

When you do a BUILD on a file or a key, if you first set PROP:ProgressEvents Clarion will send events, including the following, back to your accept loop as the BUILD progresses.

```
EVENT:BuildFile   EQUATE ( 240H)
EVENT:BuildKey    EQUATE ( 241H)
EVENT:BuildDone   EQUATE ( 242H)
```

Windows events are really just numbers, and within the range of all possible event numbers those from 400H to 0FFFH are available for the programmer's own use. You can POST() these events and they will appear in the target ACCEPT loop. It's common to see events defined as EVENT:User + n where n is any number (with the result not exceeding EVENT:Last).

```
EVENT:User        EQUATE ( 400H)
EVENT:Last        EQUATE ( 0FFFH)
```

Standard equates

The STD equates define Windows standard behavior which you can associate with Menu items, by placing the corresponding STD: attribute in the Std ID field in the menu editor (there is a droplist of equates as well).

```
STD:WindowList    EQUATE ( 1 )
STD:TileWindow    EQUATE ( 2 )
STD:CascadeWindow EQUATE ( 3 )
STD:ArrangeIcons  EQUATE ( 4 )
STD:HelpIndex     EQUATE ( 5 )
STD:HelpOnHelp    EQUATE ( 6 )
STD:HelpSearch    EQUATE ( 7 )
STD:Help          EQUATE ( 8 )
STD:Cut           EQUATE (10 )
STD:Copy          EQUATE (11 )
STD:Paste         EQUATE (12 )
STD:Clear         EQUATE (13 )
STD:Undo          EQUATE (14 )
STD:Close         EQUATE (15 )
STD:PrintSetup    EQUATE (16 )
STD:TileHorizontal EQUATE (17 )
STD:TileVertical  EQUATE (18 )
```

Cursors and icons

There are a number of standard Windows cursors, which you can use by calling SETCURSOR(*cursor equate*). When you're done and you want to set the cursor back to the default, always call


SETCURSOR() with no parameters to turn off the temporary cursor.

CURSOR:None	EQUATE ('<0FFH,01H,00H,00H>')
CURSOR:Arrow	EQUATE ('<0FFH,01H,01H,7FH>')
CURSOR:IBeam	EQUATE ('<0FFH,01H,02H,7FH>')
CURSOR:Wait	EQUATE ('<0FFH,01H,03H,7FH>')
CURSOR:Cross	EQUATE ('<0FFH,01H,04H,7FH>')
CURSOR:UpArrow	EQUATE ('<0FFH,01H,05H,7FH>')
CURSOR:Size	EQUATE ('<0FFH,01H,81H,7FH>')
CURSOR:Icon	EQUATE ('<0FFH,01H,82H,7FH>')
CURSOR:SizeNWSE	EQUATE ('<0FFH,01H,83H,7FH>')
CURSOR:SizeNESW	EQUATE ('<0FFH,01H,84H,7FH>')
CURSOR:SizeWE	EQUATE ('<0FFH,01H,85H,7FH>')
CURSOR:SizeNS	EQUATE ('<0FFH,01H,86H,7FH>')
CURSOR:DragWE	EQUATE ('<0FFH,02H,01H,7FH>')
CURSOR:Drop	EQUATE ('<0FFH,02H,02H,7FH>')
CURSOR:NoDrop	EQUATE ('<0FFH,02H,03H,7FH>')
CURSOR:Zoom	EQUATE ('<0FFH,02H,04H,7FH>')


Here's what the cursors look like:

 **CURSOR:None**

 **CURSOR:SizeNWSE**


 **CURSOR:Arrow**

 **CURSOR:SizeNESW**

 **CURSOR:IBeam**

 **CURSOR:SizeWE**

 **CURSOR:Wait**

 **CURSOR:SizeNS**


 **CURSOR:Cross**

 **CURSOR:DragWE**


 **CURSOR:UpArrow**

 **CURSOR:Drop**

 **CURSOR:Size**

 **CURSOR:NoDrop**

 **CURSOR:Icon**

 **CURSOR:Zoom**

























As with cursors, Windows supplies a number of standard icons.

```

ICON:None           EQUATE ( '<0FFH,01H,00H,00H>' )
ICON:Application   EQUATE ( '<0FFH,01H,01H,7FH>' )
ICON:Hand          EQUATE ( '<0FFH,01H,02H,7FH>' )
ICON:Question      EQUATE ( '<0FFH,01H,03H,7FH>' )
ICON:Exclamation   EQUATE ( '<0FFH,01H,04H,7FH>' )
ICON:Asterisk      EQUATE ( '<0FFH,01H,05H,7FH>' )
ICON:Pick          EQUATE ( '<0FFH,02H,01H,7FH>' )
ICON:Save          EQUATE ( '<0FFH,02H,02H,7FH>' )
ICON:Print         EQUATE ( '<0FFH,02H,03H,7FH>' )
ICON:Paste         EQUATE ( '<0FFH,02H,04H,7FH>' )
ICON:Open          EQUATE ( '<0FFH,02H,05H,7FH>' )
ICON:New           EQUATE ( '<0FFH,02H,06H,7FH>' )
ICON:Help          EQUATE ( '<0FFH,02H,07H,7FH>' )
ICON:Cut           EQUATE ( '<0FFH,02H,08H,7FH>' )
ICON:Copy          EQUATE ( '<0FFH,02H,09H,7FH>' )
ICON:Child         EQUATE ( '<0FFH,02H,0AH,7FH>' )
ICON:Frame         EQUATE ( '<0FFH,02H,0BH,7FH>' )
ICON:Clarion       EQUATE ( '<0FFH,02H,0CH,7FH>' )
ICON:NoPrint       EQUATE ( '<0FFH,02H,0DH,7FH>' )
ICON:Zoom          EQUATE ( '<0FFH,02H,0EH,7FH>' )
ICON:NextPage      EQUATE ( '<0FFH,02H,0FH,7FH>' )
ICON:PrevPage      EQUATE ( '<0FFH,02H,10H,7FH>' )
ICON:JumpPage      EQUATE ( '<0FFH,02H,11H,7FH>' )
ICON:Thumbnail     EQUATE ( '<0FFH,02H,12H,7FH>' )
ICON:Tick          EQUATE ( '<0FFH,02H,13H,7FH>' )
ICON:Cross         EQUATE ( '<0FFH,02H,14H,7FH>' )
ICON:Connect       EQUATE ( '<0FFH,02H,15H,7FH>' )
ICON:Print1        EQUATE ( '<0FFH,02H,16H,7FH>' )
ICON:Ellipsis      EQUATE ( '<0FFH,02H,17H,7FH>' )
ICON:VCRtop        EQUATE ( '<0FFH,02H,81H,7FH>' )
ICON:VCRrewind     EQUATE ( '<0FFH,02H,82H,7FH>' )
ICON:VCRback       EQUATE ( '<0FFH,02H,83H,7FH>' )
ICON:VCRplay       EQUATE ( '<0FFH,02H,84H,7FH>' )
ICON:VCRfastforward EQUATE ( '<0FFH,02H,85H,7FH>' )
ICON:VCRbottom     EQUATE ( '<0FFH,02H,86H,7FH>' )
ICON:VCRlocate     EQUATE ( '<0FFH,02H,87H,7FH>' )

```

This is what the standard icons look like.

 ICON:None	 ICON:Frame
 ICON:Application	 ICON:Clarion
 ICON:Hand	 ICON:NoPrint
 ICON:Question	 ICON:Zoom
 ICON:Exclamation	 ICON:NextPage
 ICON:Asterisk	 ICON:PrevPage
 ICON:Pick	 ICON:JumpPage
 ICON:Save	 ICON:Thumbnail
 ICON:Print	 ICON:Tick
 ICON:Paste	 ICON:Cross
 ICON:Open	 ICON:Connect
 ICON:New	 ICON:Print1
 ICON:Help	... ICON:Ellipsis
 ICON:Cut	⏪ ICON:VCRtop
 ICON:Copy	⏮ ICON:VCRrewind
 ICON:Child	◀ ICON:VCRback

That's the first half of EQUATES.CLW – [click here for Part 2](#).

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

Reader Comments

[Add a comment](#)

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Clarion MAGAZINE

Using The TPS ODBC Driver

by Vince Du Beau

Published 2001-05-17

In this article I will explore the possibilities of using the TPS ODBC driver with other applications. You might be asking yourself, "Why do I need this?" Imagine the following scenario:

Your client is about to sign an agreement for you to build a killer application. Then you're asked, "Will I be able to use the data in Excel or Word?" You can of course explain that you need to rewrite the quote to add export capability, or you can point out the TPS ODBC driver, and show the client how to use it (billable time, of course).

The ODBC driver I'm discussing here is the developer version that comes with Clarion. It will display a notice every time you access it. Do *not* distribute this driver to clients. Your clients will need to purchase their own licenses.

Setting up the ODBC driver

In this example I will use the invoice database provided with the Clarion examples. You first need to go into the ODBC Administrator. This is usually found in the either the Control Panel or the Administrative Tools, depending on your version of Windows

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

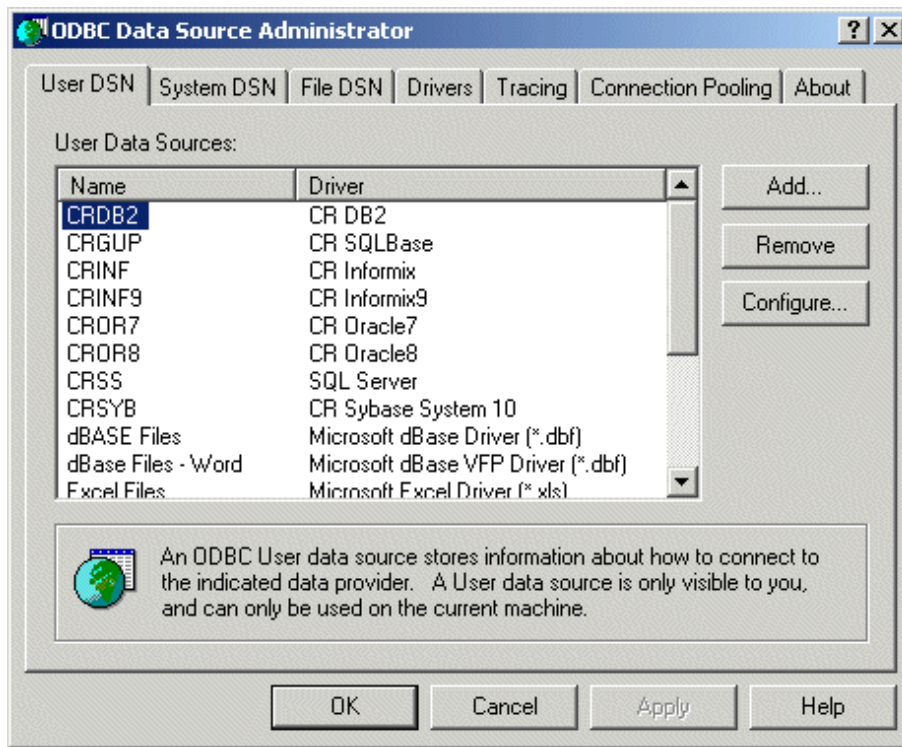


Figure 1. The ODBC Data Source Administrator, User DSN tab

On the User DSN tab, click on Add and you will be presented with the Create New Data Source dialog. Scroll down until you find the Topspeed drivers. You will find the normal driver and a read-only version. Setting this up for a client, you might want to choose the read-only version, as shown in Figure 2.

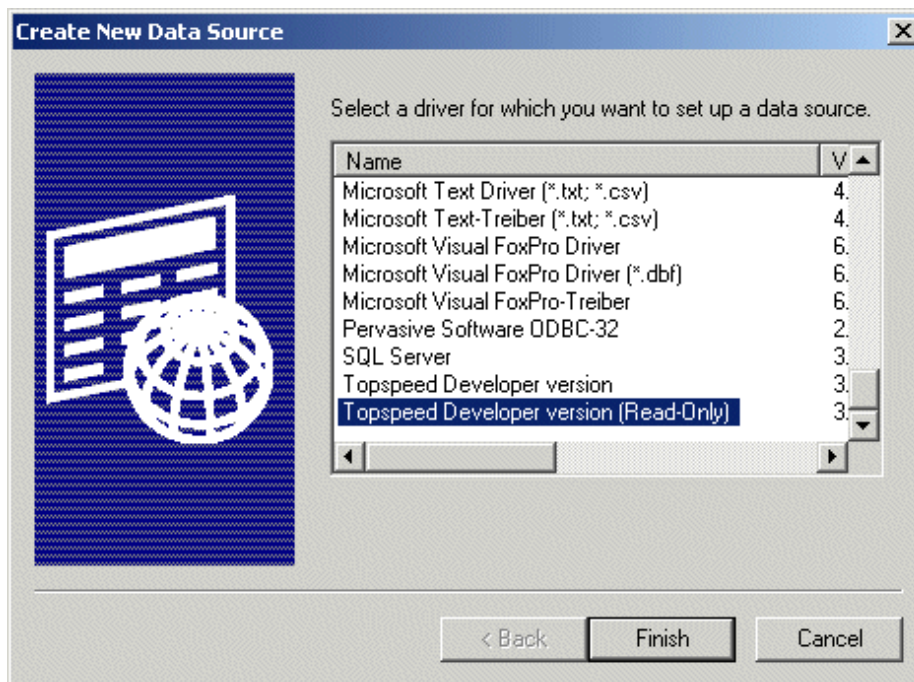


Figure 2. Creating a new data source

Click the finish button and you will see the Topspeed Data Source Name Configuration dialog. Fill the dialog in as in Figure 3, replacing the

data directory with the location of your example files.

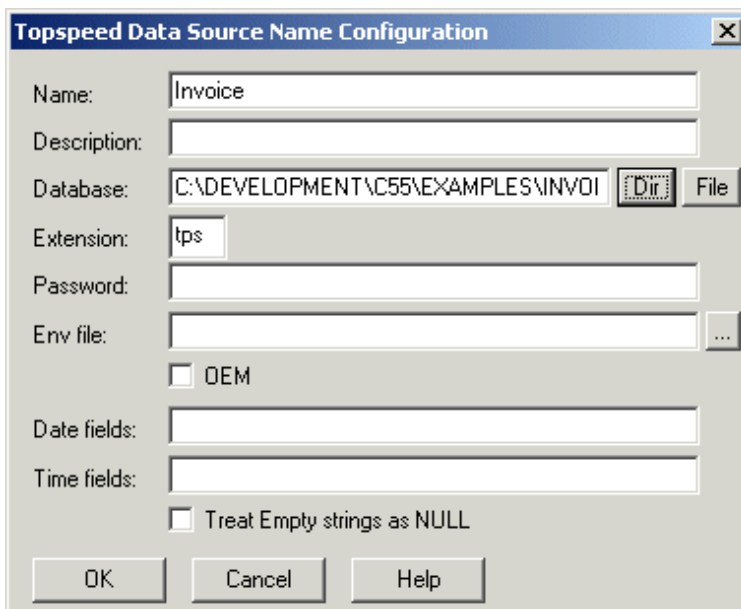


Figure 3. Configuring the TPS data source

At the bottom of the configuration dialog, you will notice fields for Date and Time. If you specify fields from your table here, the driver will convert them to ODBC compliant dates or times. You can specify single fields, multiple fields or use wildcards for field names. The online help gives more detail and also provides some hints to converting Clarion LONG dates to other applications.

Making the connection

A good way to demo the ODBC capabilities to a client is by using Excel. Here's how to set up a simple spreadsheet using fields from the Products table. After opening Excel, you have to use the Data -> Get External Data -> New Database Query menu. This will bring up the Choose Data Source dialog. Scroll down to the Invoice data source, highlight, and click OK as in Figure 4.

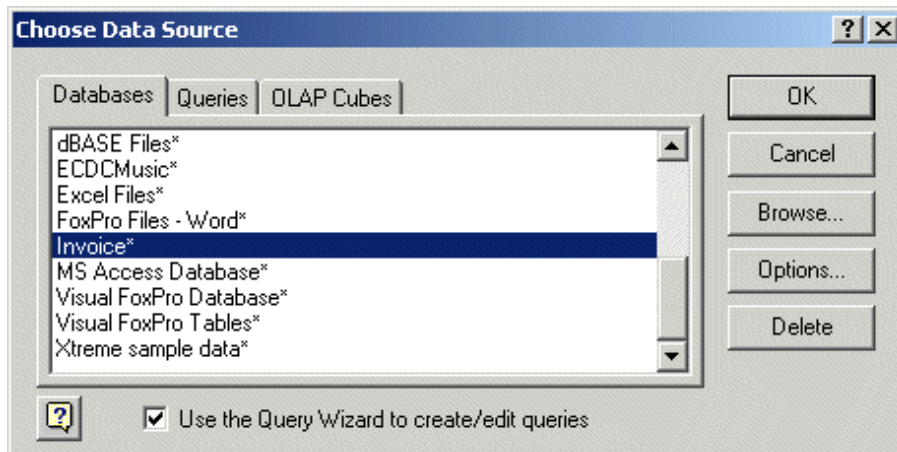


Figure 4. Choosing an ODBC data source in Excel

You will then have to choose the table and fields that you want to

use for your query. For the demo, I've chosen fields as shown in Figure 5.

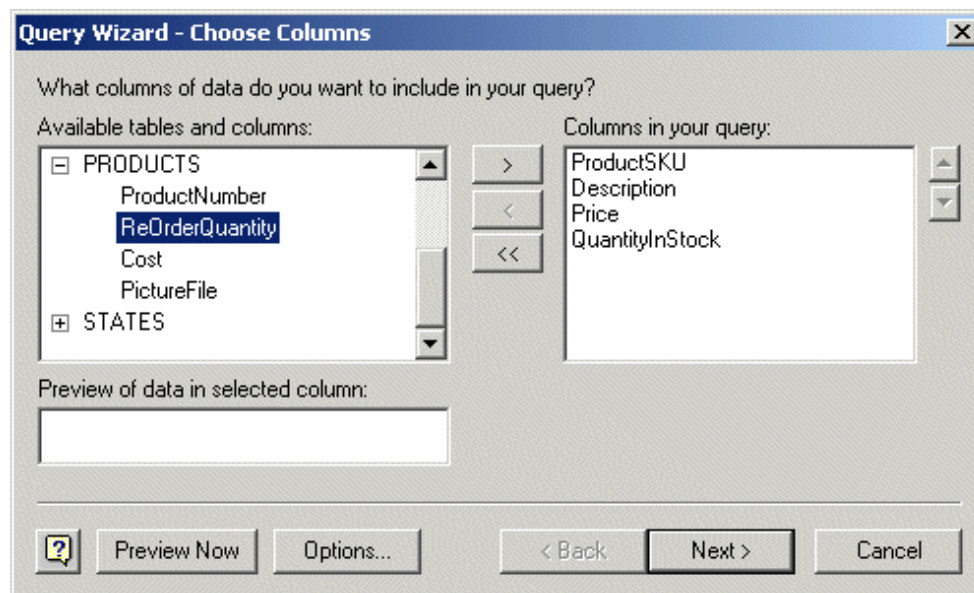


Figure 5. Selecting fields to import

The next two screens allow you to set filters and sort order. Click Next, Next, and Finish to skip through these and finish the process. The query will show a Returning Data to Microsoft Excel dialog. This dialog will let specify where you want the data returned in the spreadsheet. The properties button is what you will select. This brings up the External Data Range Properties dialog. You can play around with the options but for now I'll set it up as in Figure 6.

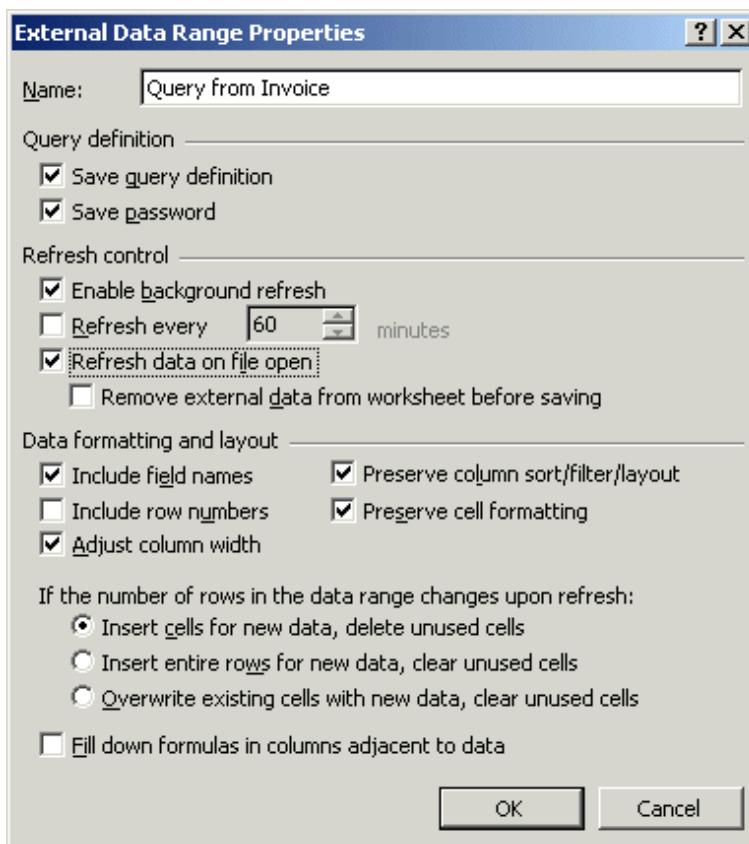


Figure 6. The External Data Range Properties dialog

Click OK, OK and you will see the Product data in the spreadsheet. Having checked the "Refresh data on file open" box, the spreadsheet will reflect any changes made to the data when it is open.

Summary

This was a very simple demonstration of using the TPS ODBC driver. I think this driver offers a lot of potential for promoting good will with clients by showing that their data is not isolated because of this strange thing called Clarion.

[Vince Du Beau](#) is the host of the radio talk show talk Bit 'n Bytes on WALE from Providence, Rhode Island. His company, Plover Development Group Inc., does AS/400 consulting and custom PC development with Clarion.

Reader Comments

[Add a comment](#)

Vince, Excellent article! And very good point about...

Vince: Nice article - you make ODBC setup simple and it...

Mac - It is installed when you install C5.5EE. Just look...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

Loading DLLs At Runtime - Part 1

by Larry Sand

Published 2001-05-16

Do you want to sell your software with optional modules that are automatically recognized when installed? Do you ever need to call a procedure that may not exist on your end user's system? Many Windows API functions are only available when certain versions of Internet Explorer are installed or your program is running on a specific version of Windows. Will your program even load if you use one of those functions?

When your application implicitly links a function in a DLL to your program, and that DLL does not exist on the end users system, your program will not load. When your user sees a run-time halt with a message something like "The mydll.dll could not be found", they will not be impressed. Even more insidious, what if the user's system has the DLL but it's an older version that doesn't have the procedure that you need? How can you avoid this kind of error? Run-time dynamic linking is the answer.

There are two ways to call a procedure or function in a DLL: implicit or load-time dynamic linking, and explicit or run-time dynamic linking. You are already familiar with the first method as it's the one Clarion normally uses. I'll cover this briefly, then explain how you can use run-time dynamic linking to your advantage in 32-bit applications.

Libraries, Libraries, and more Libraries

In Windows programming, the word "library" can cause confusion because it can mean different things. There are three kinds of libraries:

- Dynamic link libraries (such as .DLL, .EXE, and Others)
- Object code libraries (.LIB)
- Import link libraries (.LIB).

I'll assume you're already familiar with dynamic link libraries (for more information, see [Russ Eggen's article](#) in the COL archives. Object code libraries are blocks of object code that you statically link into your program. You create object code libraries in Clarion when you compile an application with the target type set to Library. The

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

last type, import link libraries, contain information that the linker uses to resolve imported procedures' addresses and references to object code modules.

Load-time dynamic linking requires that your program know where a procedure's code is located at in the DLL. This is the job of the import link library (.LIB). When you compile an application, Clarion creates an import library with the .LIB file extension. Each of these import library files contains the information that the linker uses to resolve the addresses of exported procedures and a list of required .OBJ files. Since EXEs normally don't export procedures, their libraries only contain references to object code modules. You can also use the Clarion LibMaker utility to create import libraries for 3rd party DLLs or Windows API functions not included in Clarion's import library Win32.LIB.

When Windows loads a program, it also loads all load-time dynamically linked DLLs used by that program. For programs with many DLLs and hundreds of exported procedures, this can take a significant amount of time. If your program has modules for data import or export, or specialized reporting that are not usually called, , you can reduce the load time and virtual memory requirements by using run-time dynamic linking.

Run-time Dynamic Linking

Using run-time dynamic linking, your program only loads and maps the DLL into its address space when you need to use one of its procedures. You use the Windows API functions LoadLibrary, GetProcAddress and FreeLibrary to perform this run-time linking. You also use LoadLibrary to load some specialized dynamic link libraries have extensions other than .DLL. However, this article assumes that the library is a DLL.

Function pointers

In a newsgroup message, James Harper described a method of using the DLL and NAME attributes on a procedure prototype to call a procedure by address. Before his post, I always used a TopSpeed C module to declare a function pointer and call the procedure. This required quite a bit of extra typing, and remembering the syntax for the C declaration is a pain. Consider the following C code, which declares a function type, a function pointer, and then makes the function call.

```
typedef long(far pascal fnptr)(long myLong);
typedef fnptr far* LPFUNC;
long pascal far CfLrL (fnptr fp, long myLong)
{
    return(((LPFUNC)fp)(myLong));
}
```

This C code calls any function that takes a long as its first parameter and returns a long. The address stored in the variable fp controls which function executes.

Then in your Clarion map, you prototype this just like any other external function.

```
MAP
MODULE('SomeExternalModule')
...
MyFunc(LONG FnPtr, LONG int1),LONG, PASCAL, NAME('CfLrL')
END
END
```

The only requirement is that you assign the address of the procedure to `FnPtr` before you call `MyFunc`. More on that later.

In contrast, James Harper's method dispenses with the C code by using the fact that the `DLL` attribute on a prototype in 32 bit causes the compiler to dereference a pointer. The online help for the `DLL` attribute states: "The `DLL` attribute is required for 32-bit applications because `.DLLs` are relocatable in a 32-bit flat address space, which requires one extra dereference by the compiler to address the variable." It is this one extra dereference that means the compiler expects to find the address of the procedure in a variable.

Consider the following prototype and function pointer variable declaration.

Listing 1. Prototyping an external function.

```
MAP
MODULE('SomeExternalModule')
...
MyFunc(LONG int1),LONG, PASCAL, DLL(_fp_)
END
END
fpMyFunc          LONG,NAME('MyFunc')
```

This procedure accepts a long integer as its parameter and returns another long integer, just like the previous C style function prototype. The differences between this prototype and the previous are the lack of the `FnPtr` parameter, `NAME` attribute, and the addition of `DLL` attribute.

Have you ever noticed the `DLL` attribute of a procedure coded as `DLL(dll_mode)` in generated code? If you search through the generated code for the definition of `dll_mode` you'll never find it. The template writer made use of the fact that the `DLL` attribute is considered active when the flag parameter is anything other than zero (0). It is legal to use an undefined label like `dll_mode` to turn the attribute on.

This article and accompanying classes use `_fp_` notation as the `DLL` flag attribute for a function pointer. This undefined label turns the `DLL` attribute on. It is the same as using `DLL(1)`. The `_fp_` highlights the fact that the prototype expects a function pointer variable to contain the address of the procedure/function. Here's the key to how this technique works. The `NAME('MyFunc')` attribute of the variable

`fpMyFunc` instructs the compiler to refer to the `fpMyFunc` variable whenever it encounters a call to `MyFunc`. Because all of the symbols' addresses are resolved by this circular reference, the linker is also happy and doesn't complain. Before you invoke any procedure declared this way, you *must* assign the valid address of a procedure with a matching prototype to the function pointer variable (`fpMyFunc` in this example):

```
fpMyFunc = <address determined at runtime>
Result = MyFunc(456)
```

Failure to follow this rule will get you an unpleasant visit from Dr. Watson! Furthermore, leaving the `DLL(_fp_)` attribute off the procedure prototype causes the compiler to use the address of the `fpMyFunc` variable, when what you want is for the compiler to use the address *contained in* the `fpMyFunc` variable. This will earn you another house call from the doctor.

Need proof? Symbols, assembler mnemonics, and hexadecimal, oh my!

Assume that you have a program that contains the declarations in Listing 1. Say that your program contains the following code to initialize the function pointer and call `MyFunc`. Note the fictional address used in place of `GetProcAddress` (this is only for illustration).

```
fpMyFunc = 123456h !Use GetProcAddress()
Result = MyFunc(456)
```

Note: Don't attempt to run this code, it will GPF.

If you compile this code, start the debugger, and view the disassembly you would see this assembly code starting at the `fpMyFunc` assignment:

```
mov    dword [MyFunc], 00123456
push  000001C8
call  dword [MyFunc]
mov   [RESULT], eax
```

The first line stores `00123456` hex (the fictional address) to the memory pointed to by the `MyFunc` symbol. Why does it refer to the symbol of the procedure prototype instead of the symbol for the `fpMyFunc` variable? The answer is that since the `fpMyFunc` variable declaration uses the `NAME` attribute, `fpMyFunc` refers to the procedure's symbol `MyFunc`.

Line 2 pushes the value `01C8` hex (456 decimal) onto the stack for the procedure's only parameter. The third line executes the procedure's code starting at the address contained in the memory at `MyFunc`. Remember that `fpMyFunc` and `MyFunc` are now the same symbol (they are the same address), so the code executed is located at the address assigned to `fpMyFunc`.

Finally, the result, returned in the 32-bit register `eax`, is stored in the

Result variable. If you forget the `DLL(_fp_)` attribute on the prototype, you will see this assembly language instead:

```
mov    dword [MyFunc], 00123456
push  000001C8
call   near MyFunc
mov    [RESULT], eax
```

Do you see the difference? The third line now attempts to execute the code located at the address of the `MyFunc` symbol instead of the contents of `MyFunc`. The problem is that there's no code at that address; it is the address of the function pointer variable. The square brackets around a symbol instruct the assembler to refer to the contents of the memory (a pointer), not the symbol itself. Furthermore, this is a near call instead of a far call. A near call means that the code is located within 64KB of the instruction.

Load Library Class

Now that you have a simple way to call a procedure using its address (via a function pointer) , you only need to have a mechanism to find the address to assign to the function pointer variable. This is where the load library class comes into play, and that's where I'll resume [next week](#).

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](#)

Excellent article! I already have a "load library" class...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free

CLARION
online

published by
CoveComm Inc.

Clarion MAGAZINE

The Clarion Advisor: Avoiding GPFs With ANYs And QUEUES

by Dave Harms and Jeff Slarve

Published 2001-05-10

If you've ever used an ANY variable in a QUEUE, chances are you've encountered at least one GPF. ANYs in QUEUES require some special handling, and unfortunately the Clarion documentation is not completely accurate. The LRM states the following:

- You must either CLEAR the QUEUE structure or reference assign NULL to the ANY variable (AnyVar &= NULL) before adding a new QUEUE entry.

The documentation further explains what to do when you delete a record from a QUEUE which contains an ANY:

- You must either CLEAR the QUEUE structure, or reference assign NULL to the ANY variable (AnyVar &= NULL), before deleting the QUEUE entry.

Both of these statements suggest that CLEAR(q) and q.anyvar &= NULL are equivalent, but this is not the case, as pointed out by Alexey Solovjev in several newsgroup messages. In short, Alexey's recommendation is as follows:

- Use CLEAR before you add a record to a QUEUE.
- Use &= NULL before you delete a record from a QUEUE.

CLEAR and &= NULL are mutually exclusive statements, in this usage. If you CLEAR the QUEUE or a field in the QUEUE, CLEAR simply clears a space inside the QUEUE buffer for all fields in the QUEUE, or just the field specified. CLEAR will not dispose of any object already assigned to the QUEUE's ANY variable. The &= NULL assignment, however, clears space for that variable and also destroys any previously assigned ANY, and that's the key to the majority of ANY GPFs. After you add a record to the QUEUE, whatever you just added is still in the QUEUE buffer. If you then do a NULL assignment:

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

```
q.anyvar &= NULL
```

you are actually destroying the object you previously created. You have to do a `CLEAR` to empty the `QUEUE` buffer and create space for the record you're about to add, and *then* you can do a reference assignment on the `ANY` variable in the `QUEUE`.

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

[Jeff Starve](#) is an independent software developer and the creator of the critically-acclaimed [In Back](#) automated file safeguard utility. Jeff has been a Clarion developer since 1991, and is a member of the group formerly known as Team TopSpeed.

Reader Comments

[Add a comment](#)

[FWIW - If you issue Q.Field &= NULL to clear a field and...](#)

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Clarion MAGAZINE

Quickbooks-Style Date Fields

by **Andrew Guidroz II**

Published 2001-05-10

This week, a client asked for an interesting feature. On an invoice screen, this client often times "bumps" the due date just a bit in order to make the terms of payment easier on the customer. The users wanted an easier way to do this than keying in a new date. I thought, "spin box" and changed the entry control. But this wasn't what they were looking for either.

The bookkeeper there uses QuickBooks™. There is a feature in QuickBooks™ that allows the end user to press the + key to increase the date by a day or press the – key to decrease the date by a day. This should be pretty easy. Or so I thought.

Step one was to add alert keys to the field. Now, which keys are those? I came up with 189 as the keycode for the minus sign and 443 for the plus sign. Everything looked great and, within the IDE, I alerted those two keys. Then, in the embed for `EVENT:AlertKey` for that particular control, I wrote the following:

```
IF Loc:DateField <> 0
  IF KeyCode() = 443
    Loc:DateField += 1
  ELSIF KeyCode() = 189
    Loc:DateField -= 1
  END
  DISPLAY(Loc:DateField)
END ! Loc:DateField <> 0
```

And that was that. Or so I thought.

The bookkeeper asked me why it still didn't work. I walked over and tested it and it seemed to work just fine. Then the bookkeeper used the + and – keys on the numeric keypad. "DOH!" So I added code to account for those keys.

```
IF Loc:DateField <> 0 |
  AND ?Loc:DateField{PROP:ReadOnly} <> TRUE
  IF KeyCode() = 443 OR KeyCode() = PlusKey
    Loc:DateField += 1
```

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)


```

    ELSIF KeyCode() = 189 OR KeyCode() = MinusKey
        Loc:DateField -= 1
    END
    DISPLAY(?Loc:DateField)
END ! Loc:DateField <> 0

```

And everything worked great. The bookkeeper loved it. And then the next call came.

"The feature doesn't work in the due date field on the other invoicing system." Well, of course it doesn't. I had to modify that field also. And then I realized a template was about to happen.

First, how was I going to alert the keycodes programmatically rather than within the IDE? After reading a bit, I found out that Alert keys are stored as an array within Clarion. If you specify the highest element of the array, which is 255, then the runtime will automatically push it down to the first available entry in the array. That makes life smoother. So at the end of the `WindowManager Init` embed points, priority 9001, I wrote the following:

```

?Loc:DateField{PROP:Alrt,255} = 443
?Loc:DateField{PROP:Alrt,255} = 189
?Loc:DateField{PROP:Alrt,255} = PlusKey
?Loc:DateField{PROP:Alrt,255} = MinusKey

```

Now, the next thing I needed was the code to react to the event to be in an embed point that is less specific to the individual control. That seemed to be the `WindowManager TakeEvent` embed point at priority 6300. And the code I added looked like this:

```

CASE EVENT()
OF EVENT:AlertKey
    CASE Field()
    OF ?Loc:DateField
        IF Loc:DateField <> 0 AND |
            ?Loc:DateField{PROP:ReadOnly} <> TRUE
            IF KeyCode() = 443 OR KeyCode() = PlusKey
                Loc:DateField += 1
            ELSIF KeyCode() = 189 OR KeyCode() = MinusKey
                Loc:DateField -= 1
            END
            DISPLAY(?Loc:DateField)
        END ! Loc:DateField <> 0
    END ! Case Field()
END ! Case EVENT()

```

Now, my code was shaping up. It was template time.

A control template was not going to cut it because that would require repopulating my fields. What I wanted was an extension template to add to a window that would give any date field on that window this functionality. So, here comes the template. (NOTE: The following template has some line breaks added so it will fit on this page. For the actual template, see the download link at the

end of the article.)

```
#EXTENSION(QuickenDateAndTime,↵
    'This adds Quicken Style Date fields. '),↵
    DESCRIPTION('This adds Quicken Style ↵
    Date fields. '),PROCEDURE
#DISPLAY('This will work for entry ↵
    fields with an @D in them. ')
#ATSTART
#DECLARE(%MyControls),MULTI,UNIQUE
#FOR(%CONTROL),WHERE((%ControlType = ↵
    'ENTRY' OR %ControlType = 'SPIN') ↵
    AND ((INSTRING('@D',↵
    UPPER(%ControlStatement),1,1) <> 0)))
    #ADD(%MyControls,%Control)
#ENDFOR
#ENDAT
#AT(%WindowManagerMethodCodeSection,'Init'↵
    ,'(),BYTE'),PRIORITY(9001)
#IF(%MyControls)
    ! The following alerts are for
    ! Quicken Style Date Fields
#FOR(%MyControls)
    #FIX(%Control,%MyControls)
    %CONTROL{Prop:Alrt,255} = 443
    %CONTROL{Prop:Alrt,255} = 189
    %CONTROL{Prop:Alrt,255} = PlusKey
    %CONTROL{Prop:Alrt,255} = MinusKey
#ENDFOR
#ENDIF
#ENDAT
#AT(%WindowManagerMethodCodeSection,↵
    'TakeEvent','(),BYTE'),PRIORITY(6300)
#IF(%MyControls)
    ! The following code handles
    ! the alert keys for Quicken Style Date Fields
    CASE EVENT()
        OF EVENT:AlertKey
            CASE Field()
                #FOR(%MyControls)
                    #FIX(%Control,%MyControls)
                    OF %Control
                        IF %ControlUse <> 0 |
                            AND %Control{PROP:ReadOnly} <> TRUE
                            IF KeyCode() = 443 |
                                OR KeyCode() = PlusKey! Pluskey
                                    %ControlUse += 1
                                ELSIF KeyCode() = 189 |
                                    OR KeyCode() = MinusKey
                                        %ControlUse -= 1
                                END
                                DISPLAY(%Control)
                            END ! %ControlUse <> 0
                        #ENDFOR
                    END ! Case Field()
```

```

    END ! Case EVENT()
#ENDIF
#ENDAT

```

The #ATSTART code creates a local variable called %MyControls that will be used to store all fields on the window that are date fields. Dates can be entered in ENTRY and SPIN controls. Also, dates contain @D in the control picture, so it's a simple matter to loop through all the controls on a window and identify the dates.

The code in the WindowManager Init area places alert keys on each control. I could also alert (and add code for) the other Quicken™/Quickbooks™ date hotkeys (such as w for the first day of the week, k for the last day of the week, etc).

The code in the WindowManager TakeEvent area creates a CASE statement for each control, if any are present. Everything is pretty easy so far. But how do I make this happen throughout my entire program?

I began to write a template that looped through every procedure and could be added to the global extensions. But I hit a wall almost immediately with it. Then, I noticed the Application attribute for an extension template. The documentation says

APPLICATION Tells the Application Generator to make the #EXTENSION available only at the global level.

child(chain) The name of a #EXTENSION with the PROCEDURE attribute to automatically populate into every generated procedure when the #EXTENSION with the APPLICATION attribute is populated.

This sounded exactly like what I needed. So, I wrote one other small extension template...

```

#EXTENSION(QuickenDate, ␣
    'This adds Quicken Style Date fields.')(␣
    ,DESCRIPTION('This adds Quicken Style Date fields.')(␣
    ,APPLICATION(QuickenDateAndTime)
#DISPLAY('This will work for entry fields ')
#DISPLAY(' with an @D in them.')

```

And that's it. I add this extension template to the global extensions for every app I have and it automatically adds the QuickenDateAndTime procedure extension to every procedure. The code only gets generated for those windows that contain date entry or spin controls.

And, after a recompile, the bookkeeper is happy, I'm happy, and the editor at Clarion Mag has a new article.

[Download the source](#)

Andrew Guidroz II, when he isn't handfeeding the tufted titmouse, writes software for all facets of the insurance industry. His famous Cajun cookouts have become a central feature of Clarion conferences throughout the U.S. Andrew's Cajun website is www.coonass.com.

Reader Comments

[Add a comment](#)

Added to cw5.5d PE ABC, date increments by two days not...

Might the feature instead be added by extending the...

In reponse to Doug Johnson... Yup. That was another way...

To Gregg Matteson ... I just added the template directly...

Andrew, In your copious spare time, you might also add...

Clarion Online...

Andrew, While you work here is just fine, ...

In response to Mark... I wish I would have had more time...

to Lew: Yeah ... after I wrote it somebody told me there...

To Alan ... Oops . Looks like both the editor and...

Andrew.. Nice article....

Hi Andrew, Nice article. I did a similar thing in my...

Very nice article. I added additional items and brought it...

To Bruce Johnson ... That's exactly what I thought the...

To W B McDowell III: Humorous? I thought I was serious...

Great Info here, I used some of this information to alter...

To Mike ... Exactly. I really wish I had thought longer...

Didn't you read Bruce's article on CASE? Below is the...

Template Writer Utility (TWriter.EXE) would be good for...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

Creating ODBC Data Sources At Runtime

by Jon Waterhouse

Published 2001-05-09

One of the drivers that comes with Clarion is the ODBC driver. Although Clarion deals with most of the problems of translating your file access code (e.g. OPEN, CLOSE, NEXT) into calls to the particular ODBC driver that looks after your data file, there is one area where Clarion ignores a potentially useful set of features of the ODBC design. These are the administration functions, which are required before you can access any data source through ODBC.

When you are using most of the file drivers, all you need to know is the file name. When you use ODBC, however, you specify the "file" as a "Data Source Name" (DSN). The DSN contains the information that points to a specific file or directory. In this article I'll show you how to use the administration functions to create an ODBC DSN at runtime.

The problem

Imagine you have been sent one Access database for each of thirty towns, where all of the databases have the same structure. Your job is to amalgamate all of the data into one big file. If you were dealing with flat files you would probably just loop through all of the files you had to deal with. With the ODBC connection you have to have a DSN set up for each file you want to use. This means one of two things; either you manually set up (in ODBC sources in the Control Panel) all of your DSNs before you start, or you create the DSNs dynamically as you need them. As a long-term strategy, the second solution definitely sounds better to me.

There are of course, several ways to skin this cat. In ODBC each driver presents a standardized interface for a particular data source. The ODBC DLLs that come with Windows deal with adding new ODBC drivers, and pass data requests to the relevant driver. You therefore have the choice of talking to the Windows ODBC manager, or directly to the ODBC driver for the file you are interested in (assuming you have documentation for that driver). The Clarion ODBC interface approach is to talk to the ODBC manager.

NOTE: There is an ODBC back-end driver for TopSpeed files, but that's not what I'm talking about here.

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

The main procedure in the ODBC admin DLL that deals with setting up DSNs is called `SQLConfigDataSource`. This procedure calls a procedure in each particular ODBC driver called `ConfigDSN`. I'll look at calling `SQLConfigDataSource`, because it is more general than `ConfigDSN`, and will work whether your data source is Access, SQL Server or any other database with an ODBC interface.

The documentation for `SQLConfigDataSource` gives the prototype as:

```
BOOL SQLConfigDataSource(HWND hwndparent,WORD frequest←
    ,LPCSTR lpszdriver ,LPCSTR lpszAttributes)
```

In Clarion this translates into:

```
SQLConfigDataSource PROCEDURE(ULONG ParentWindow,←
    USHORT Request,*CSTRING DriverString,←
    *CSTRING AttributeString),BYTE,RAW,PASCAL
```

In general, when using non-Clarion Windows DLLs, you can rely on the following general rules:

- Handles are ULONGs
- A word is four bytes (a USHORT in Clarion), so a DWORD (double word) is eight bytes, or a ULONG
- LP as a prefix stands for long pointer. Pointers to various types of data in procedure calls are indicated by asterisks (*) in Clarion
- The RAW attribute means that strings and groups are passed without length information. In this example RAW is not strictly necessary, but it will most often be required when using external DLLs, so you might as well get in the habit of using it
- The PASCAL attribute means that procedure parameters are passed left to right on the stack, compared to C which passes them right to left on the stack, and the default calling convention used by Clarion, which is to pass parameters using registers. The PASCAL convention is used for all Windows API calls.

The first parameter to `SQLConfigDataSource` can either be NULL (in which case your activity will happen in the background without displaying a screen to the user), or you can pass it the handle of your procedure window (`0{PROP:Handle}`). The second parameter indicates what you want to do — add, change or delete a DSN. The documentation in the help file goes as far as telling you that valid values are `ODBC_ADD_DSN`, `ODBC_CONFIG_DSN` etc. You will have to look at the C header files that come with the Microsoft Data Access Components Software Developers KIT (MDAC SDK), available free from http://www.microsoft.com/data/download_260SDK.htm, to find out that the values corresponding to these EQUATES (#defines in C) are 1, 2, etc.

The final two parameters are where you specify, respectively, the driver you want to add or configure, and details of your request.

These details include what file you want attached to the DSN (for Access) and what DSN you want to give your data source . This attributes string can take several instructions of the form *Keyword= value*. The documentation suggests that each argument should be separated from the next by a NULL character, and the string terminated with a double NULL. In practice semi-colons seem to work just as well or better.

The basic steps to use this function in your application are:

1. Build a .lib file for the ODBCINST.DLL (which you will find in Windows\system (or system32))
2. Use Application|Insert Module|External DLL to add the .lib file you just created to your application
3. Add a Procedure to your application called `SQLConfigDataSource`. Specify that it is an external procedure, and type in the prototype as given above (starting with the first parenthesis)
4. In the procedure where you want to set up your new DSN create two local data fields, say `DriverString` and `AttribString`, as `CSTRING`s. The `DriverString` should be 33 characters, while the `AttribString` should be 255 characters.
5. Write the values you desire into your two `CSTRING`s. In the driver string you should put the driver name exactly as it appears on the Drivers tab in the ODBC data sources control panel application (e.g. Microsoft Access Driver (*.mdb)). In the `AttribString` you need to put your keyword value pairs as described in `ODBCJET.HLP` file (for Microsoft data sources). The ODBC Setup Dialog Page is the most useful. For example, to set up a DSN for an Access MDB database file, you could use something like: `AttribString = 'DSN=MyDataSource;DBQ='C:\my data\AccessData.mdb''`
Don't forget to double quotes where necessary.
6. Write a call in source, e.g. `retval =SQLConfigDataSource(0,1,DriverString,AttribString)`
7. Open your file (which has the DSN and any other needed values in the `OWNER` attribute) and use normally.

I suggested that you could use a scheme like this to process a whole bunch of similar files. In theory you could create a single DSN (say *temp*), use it, make another call to `SQLConfigDataSource` to change the file the DSN points to (the `DBQ=` keyword value) and then read in your next file. In practice this doesn't work. If you reuse the DSN you will keep on being connected to the first data source. Therefore, you have to create a series of DSNs (*temp1*, *temp2*, etc.) and delete each (`ODBC_DELETE_DSN=3`) as you finish with it.

That's the basics. This scheme is demonstrated in the example program at the end of this article, which simply creates several DSNs on the fly.

The next step is to make the whole process of integration a little easier. Step 1 is required, because Clarion needs the .lib file in order to link in the ODBCINST.DLL. You can't do without Step 5, either; you have to specify the driver and the associated data source.

However, steps 2,3,4 and 6 are potential candidates for a template; actually two templates. The first template is a global extension and declares the ODBC procedures in the global map. The second is a CODE template which declares the two local variables and prompts you for the driver and attribute strings. The templates can also wrap some extra code around the plain vanilla call to `SQLConfigDataSource` to get it to provide better error checking.

The `SQLConfigDataSource` procedure returns a 1 (success) or 0 (failure). There is also a function called `SQLInstallerError` that can be called to give more information about errors. I've written a procedure called `CDSWrapper` that first calls `SQLConfigDataSource`, and then reports the particular error type if the function fails.

The global extension part of the template does four things:

1. Adds the ODBC library to the project
2. Adds the prototypes for `CDSWrapper` and all of the ODBC procedures to be used to the map
3. Declares a bunch of `EQUATES`
4. Generates the code for the `CDSWrapper` procedure

From a template writing perspective, the first two items are very useful things to know how to do. You can borrow from the techniques in this template to add outside libraries and global procedures to your applications.

Adding the library to the project is accomplished by the `#PROJECT` statement. This statement has the same effect as manually doing an `Application|Insert module`.

The `%CustomGlobalDeclarations` embed point is also the place to declare the `CDSWrapper` procedure and its prototype. This requires three lines of code.

```
#ADD(%CustomGlobalMapModule,%Application & '.clw')
#ADD(%CustomGlobalMapProcedure,'CDSWrapper')
SET(%CustomGlobalMapProcedurePrototype,'←
    (BYTE,BYTE,*CSTRING,*CSTRING),BYTE')
```

The first line says that the procedure is in is the `%Application.clw` file. The second has the procedure's name, and the third declares the prototype. This is all that is needed to add the `CDSWrapper` procedure to the map, but the application will still need prototypes for all of the required procedures in the ODBC DLL. I have been lazy and hard-coded the name of the DLL(LIB) file. If I was a bit more professional about it I would extract the DLL name from the name of the LIB file, which is stored in the `%ODBClib` token.

The `EQUATES` are taken from the ODBC documentation that comes with the MDAC software developers kit. They are used in the `CDSWrapper` procedure. A number of the equates declare certain data types and handles to be equivalent to other data types. For example, `HENV` (an environment handle) is actually stored in a `Clarion ULONG`.

The `CDSWrapper` procedure is placed in the `%ProgramProcedures` embed point and will therefore show up at the end of your `%Application.clw` file. The procedure is a bit more elaborate than it really needs to be, but this should give you a good start if you decide you would like to use more of the administrative ODBC functions.

The first line of code:

```
whandle = CHOOSE(display,0{PROP:Handle},0)
```

sets the window handle to be either 0 (if the code template specified no display), or to the handle of the current window (`0{PROP:handle}`). The next section of code is not really necessary. What it does is load the names of all of the available ODBC drivers on the machine into a queue. For reasons I don't claim to understand, this requires first setting up an "environment", which is what the calls to `SQLAllocHandle` and `SQLSetEnvAttr` do. The driver names are then retrieved in a loop that looks like this:

```
Get first driver
Loop
  If error
    Report error
    Break
  End !if
  Add driver to queue
  Get next driver
End ! loop
```

I mention this explicitly, because this type of loop structure is not very common in Clarion. This loop could be used to build a list of available drivers to be presented to the user.

The environment handle is then freed because it is not needed any more. The next section requires that at least one driver was found in the first part of the procedure. The driver name is checked against the list of drivers that are stored in the queue. If the driver is not there you are not going to be very successful setting up a data source that relies on that driver, so you get an error message and exit the procedure. If everything is still okay the procedure makes the call to `SQLConfigDataSource`. If `SQLConfigDataSource` returns an error, `SQLInstallerError` provides more information. `SQLInstallerError` does display a reasonably informative error message if you call `SQLConfigDataSource` with a non-existent driver name.

As I mentioned, the procedure is really more elaborate than it needs to be. The major value of the first part is that the `SQLDataSources` procedure takes exactly the same arguments as `SQLDrivers`. Thus if you want a list of all the currently available DSNs in a queue that you can display, a very modest adjustment to the first part of this procedure will give it to you. The `CDSWrapper` procedure returns either 0 (failure) or 1 (success).

The second part of the template is a code template that can be easily added to a procedure to set up a DSN. It translates into three lines of code: one to set up the driver string (based on programmer input), one to set the attribute string, and a third to call `CDSWrapper` with the appropriate parameters. The template also sets up three data fields each time you add the template (a return value (`CDSretvalue`), a driver string and an attribute string).

The four parameters required are:

1. Display the dialog box or not
2. Type of action (add, delete, user DSN or system DSN)
3. the driver string
4. the attribute string

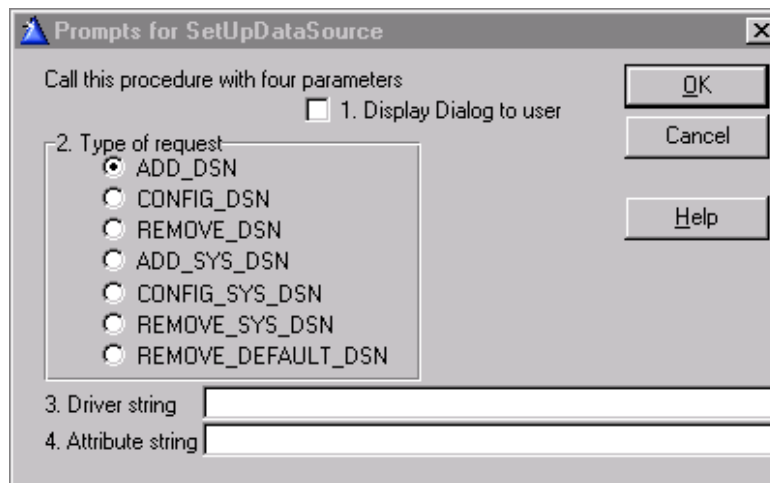


Figure 1. The SetupDataSource prompts

The example application does very little; it simply sets up two data sources. One is an Access data source (you will have to find an access (.mdb) file on your machine to reference in the DBQ field, which is set up without a dialog displayed). The other is a TopSpeed data source (for which you need the TopSpeed ODBC driver, included in CW5.5, but a separate purchase for earlier versions). This is called with a dialog displayed, mostly because it does not work otherwise. It returns a success code, but unfortunately does not set up the data source. I have version 2 of the ODBC driver, and this may be fixed in version 3.

Keep in mind that the template does not do the first step for you: you will need to make the .LIB file from ODBCINST.DLL before you use it. Also, the template can only set up data sources for drivers that are set up on the machine. If you don't have the TPS driver (read only) or the Microsoft Access driver on your machine, you'll run into errors, or you'll need to modify the example to work with data sources you do have.

Summary

Learning how to use the ODBC administrative functions has certainly made my life easier. The government department I work for has two

applications that write to local databases; one database for each regional office. Every six months or so I get two CD-ROMs containing the data uploaded from each of the twenty or so offices. Previously I would have to manually set up forty DSNs (twenty offices times two data files) to read in the data to amalgamate into a single provincial data base. Now, I just have to provide my program with the names of the main directory and the subdirectories in which the data sits, and the program can create a temporary DSN for each office data base in turn.

For me, being able to use ODBC administrator functions just saves time. However, it is not hard to think of situations where you have to write a program that incorporates a legacy data source. If you can incorporate this data without having to train users to set up DSNs on their machines you will probably save yourself a lot of headaches.

[Download the source](#)

[Jon Waterhouse](#) has been using Clarion since the 2.1 days. His main work is as an economist, and he finds that Clarion is well-suited for applications which impose order on various sets of data. His projects include questionnaire data entry programs, classification software (assigning projects to groups), plus some more interesting scheduling applications. Jon has also used Clarion to link text information together, and is currently developing a program that will store linked snippets of WordPerfect documents and print custom documents composed of several of these snippets. He is currently working for the Newfoundland Government on a project to measure the performance of government employment programs.

Reader Comments

[Add a comment](#)

A note on WORD and DWORD: The bullet point should say:

A...

Surprise !!! My system can't find the ODBCINST.DLL, but...

I believe ODBCINST is installed only with the 16bit version...

Actually i can't find the odbinst.dll on my mahcine,...

Different versions of windows keep the same functions in...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

Reader Comments Now Available On All Articles

Published 2001-05-08

You can now add your own comments to any ClarionMag or COL Archive article. At the bottom of each article you'll see an "Add a comment" link, as well as a list of any comments already posted. Just follow the links to add your own comment or read what others have to say.

Your comments will appear in plain text, so if you type in some HTML you'll see the tags, not the effect of the tags, i.e. ``this text will not be bold``. I've disabled HTML to make it easier to post Clarion code snippets, which often contain angle brackets.

My thanks to everyone who participated in the beta test. If you have any thoughts on how this feature can be improved, please email dharms@clarionmag.com

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

Reader Comments

[Add a comment](#)**I'm out of the office for a few days, so if you email me...****Sure wish Clarion mag would support Klingon fonts!****Do next and previous work?****Next and previous do work but only if there's something...****Hi Dave, Great feature! I like it! However the biggest...****There are two ways to handle paragraph breaks. One is to...****The article comments code now has improved formatting,****i.e....****You also now get a button back to the article after you...****Hi Dave. This is a nice feature, and makes Clarion...****Testing only, pls. ignore (g)**

I've made some further changes (hopefully improvements) to...

This new feature sure seems interesting. I am typing...

It would be useful to be able to see a page with details of...

I'm already working on it.

Wow, thanks - that will be very handy. (My) memory is like...

Dave, One suggestion (or request for a feature please...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Clarion MAGAZINE

Replicating IDLE: All Quiet on the Keyboard?

by **Steven Parker**

Published 2001-05-03

The Language Reference Manual states the following:

IDLE Arms a procedure that periodically executes.

An IDLE procedure is active while ASK or ACCEPT are *waiting* for user input. Only one IDLE procedure may be active at a time. Naming a new IDLE procedure overrides the previous one. An IDLE statement with no parameters disarms the IDLE process. (Emphasis added.)

I don't think that this definition of IDLE has changed since I first started using Clarion late in the last century. And this description is just as abstruse now as it was then.

Looking at the terms helps. "ASK reads a single keystroke from the keyboard buffer. Program execution stops to wait for a keystroke." Ok, Ask waits for a keystroke. And, Accept is the event handler. Therefore, in plain language, IDLE is an inactivity timer. When there has been no keyboard activity for the period of time specified in the IDLE statement, the procedure named will execute.

Unfortunately, IDLE has been broken since (at least) Clarion for Windows 1.5. And, it is broken badly. For example, the documentation states "Only one IDLE procedure may be active at a time. Naming a new IDLE procedure overrides the previous one." This is true in 16 bit; false in 32 bit (new IDLE procedures in 32 bit are unlikely to execute at all) .

The documentation also states "The IDLE procedure executes on thread one (1)--the same thread as the APPLICATION frame." This is true in 32 bit, false in 16 bit (in 16 bit, an IDLE procedure only needs to be in the same module as its caller).

In my first Windows app, a 16 bit program, I had to put this code:

```
If KeyCode( )
```

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

```

Idle()
Idle(Reset_, Cfg:Tab_Timeout)
End

```

in a browse's Accept Loop embed to get the same effect I that I got in DOS from placing a simple call to `IDLE` in the After Opening Screen embed. Clearly `IDLE` is not resetting – I think this is technically described as "restarting" -- on a keystroke. (In 32 bit, this code would do nothing at all.)

In 32 bit, an `IDLE` procedure must be in the same module as the Frame and, while it responds correctly to keystrokes on the Frame, it does not respond to keystrokes in any other procedure. In other words, in 32 bit, an `IDLE` procedure is just a global timer and can't be reset from within the program in any of the ways implied in the documentation.

And, even if `IDLE` weren't broken, what if I need two inactivity timers active simultaneously? I recently found myself in exactly this position. A cash register displays a list of items that have been rung up. When the sale is completed, the list should be blanked after, say, 30 seconds (giving the cashier a chance to print a second copy without having to retrieve the transaction). At the same time, the register program should lock itself if there has been no activity for two minutes. (In the program I'm working on, both of these delays are user definable.) To do this requires two `IDLES`. However only one can be active at a time. In other words, what I need can't be done with `IDLE` (maybe it's not so bad that it's broken).

What Does IDLE Do?

Fortunately, replicating the functionality of `IDLE`, without the one-at-a-time restriction, is not very difficult. "It's easy," Marshall Brodien used to say "once *you* know the secret."

There is one caveat and that is that replacing `IDLE`'s functionality is fairly straightforward for a single procedure or two. It is not if you need a global `IDLE`, an `IDLE` active at the Application level, at least not without substantial work (probably involving posting events to the Frame's thread).

To duplicate the core functionality of `IDLE`, I need to understand what the statement does. And, what `IDLE` does is call a procedure, named in the `IDLE` statement's first parameter, when there has been no activity for the period of time specified in the `IDLE` statement's second parameter. So I need to know the amount of time and the action to take.

In "Windows-ese," `IDLE` may be described as calling a procedure when there has been no event for a specified amount of time. This immediately suggests the `TakeEvent` method as the ideal place to check whether an event has occurred. If an event occurs, then I want to restart the timing cycle. I can use the window's `TIMER`

attribute to cause a timer event to fire periodically and trigger a call to `TakeEvent` (actually, I have to use the `TIMER` attribute).

Because I intend to use virtual methods (embeds), I am not limited to calling a procedure as a template or library call usually is. I can execute whatever code I want to. Neither am I limited, as I am by `IDLE`, to procedures that do not take parameters.

However, *any* event, including a timer event, will call `TakeEvent`. I certainly do not want to restart the timing cycle if the only thing that has happened is a timer event. To avoid resetting the time when only the `Event:Timer` trips, I can check `If Event() <> Event:Timer` and eliminate timer events from the equation entirely.

On the other hand, if the window timer *does* trip, I want to check if the time has run out. Thus, I know that I will be using `ThisWindow.TakeEvent` and [Window Events | Timer](#) to implement `IDLE`-like functionality.

Idling

My actual implementation of `IDLE` functionality employs two variables. The first variable contains the default amount of inactivity time, the default timeout, and is stored in a configuration or INI file. The second is a local variable which is used in the actual counting, set and reset by the first. This allows me the flexibility of capturing the timeout in seconds or minutes and converting it to whatever unit of measure I need.

If I set the window's timer at 100 (one second) and the default time is also in seconds, I do not have to "convert" anything. However, if the default time was entered or stored in minutes, I multiply by 60 to get the number of seconds. If I set the window timer to 50 (one-half second), I would multiply a seconds variable by 2 and a minutes variable by 120.

If I am using the built-in `Clock()` function then I have to convert to 1/100th seconds to get a Clarion Standard time. Therefore,

```
1 second = 100
1 minute = 6000
```

Check the On-line Help for "Time: Standard Time" or look at my [Clarion Online articles](#) for more.

Assume that my two variables are `GLO:NumberSeconds` and `Timeout`. In `TakeEvent`, if there has been no event (or, more precisely, no non-timer event), I want to recalculate the timeout:

```
If Event() <> Event:Timer
    Timeout = Clock() + (GLO:NumberSeconds * 100)
End
```


Timeout now contains the Clarion standard time at which I need to take action. Then, in the Timer event embed, if the last event was only a timer event, I want to compare the current time to Timeout:

```
If Event() = Event:Timer
  If GLO:NumberSeconds
    !user actually does want a timeout
    If Clock() > Timeout
      !do something
    End
  End
End
End
```

This will work perfectly if the application does not run across midnight. If midnight rollover could occur, this code will cause problems if the last non-timer event is less than GLO:NumberSeconds before midnight.

Because Timeout is calculated by simple addition, it could end up with a value higher than the maximum allowed by Clarion (8,640,000). If I try to compensate by subtracting 8,640,000 whenever Timeout exceeds 864000, the time out action will trigger immediately on the next timer event.

"StarDate" to the rescue

I do not know who devised the StarDate technique but it is exceedingly ingenious. A StarDate allows a single variable, typically a Real(15,8) or Decimal (13,6), to contain both the date and the time. Therefore, two times can be compared using StarDates without worrying about midnight rollover.

A StarDate is constructed by taking the Clarion standard date and adding the time expressed as a fraction. Thus, the portion of a StarDate to the left of the decimal (the integer portion) is the date and the portion to the right (the fractional portion) is the time.

Time can be expressed as a fraction by dividing it by 8640000 (the maximum number of clock ticks in a day). For example:

```
StarDate = Today() + ( Clock() / 8640000 )
```

StarDates are converted back into a date and a time by reversing the arithmetic.

```
DateVar = Int(StarDate)
```

The time is the fractional portion and I get it by subtracting the date:

```
TimeVar = StarDate - INT(StarDate)
```

To finish the conversion to a Clarion standard time, multiply by 8640000.

```
TimeVar *= 8640000
```

The IDLE code above, adjusted to handle midnight rollover, becomes:

```
!TakeEvent
If Event() <> Event:Timer
    Timeout = Today() + |
        ((Clock() + (GLO:NumberSeconds * 100)) |
        / 8640000)
End
```

and

```
!Timer
If Event() = Event:Timer
    If GLO:NumberSeconds > 0
        !new day, adjust for midnight rollover
        If (Today() > INT(Timeout)) |
            AND (Clock() + 8640000 > |
                ((Timeout - INT(Timeout)) * 8640000)) |
            OR (Today() = INT(Timeout)) AND |
                (Clock() > ((Timeout - INT(Timeout)) * 8640000))
            !do something
        End
    End
End
```

Note: just because `Today() > INT(Timeout)` does not *ipso facto* mean that `GLO:NumberSeconds` have elapsed.

The code above works by checking the dates first then making the appropriate time comparison. Alternately, I could create a second `StarDate` in the Timer embed and subtract:

```
If Event() = Event:Timer
    If GLO:NumberSeconds
        StarDateNow = Today() + ( Clock() / 8640000 )
        If StarDateNow - Timeout > 0
            !do something
        End
    End
End
```

This seems much easier to read. (Yes, I also could have checked `StarDateNow > Timeout`.)

But wait ... there's more!

The techniques described above work. The first, of course, does have a midnight rollover restriction. Calculating the time at which

to act has the virtue of working with any timer value, from 1 to 6553 (the maximum imposed by Windows), not only one second timers. But, it seems to me that I should not really *need* to calculate the time of day at which I need to time out the procedure, at least not when I am willing to set my window timer to one second.

Since I know the number of seconds (or minutes) after which to time out, I should be able to simply count down the remaining time. Something like:

```
If Event() = Event:Timer
    Timeout -= 1
    If Timeout = 0
        !do something
    End
End
End
```

If I do this, then `TakeEvent` becomes a simple resetting `Timeout` to its initial value:

```
If Event() <> Event:Timer
    Timeout = GLO:NumberSeconds
End
```

Or

```
If Event() <> Event:Timer
    Timeout = GLO:NumberMinutes * 60
End
```

In this case, midnight rollover is automatically handled (or, more accurately, entirely ignored as time of day never enters the computation at all). The sample apps accompanying this article (one C5, one C55) implement this technique. Counting down works. (My register program implements the `StarDate` technique.)

Counting down works with one small exception (and this "small exception" also affects the first techniques discussed). If there is an MDI Frame procedure and the "Display the date and/or time in the current window" extension template is used, the timeout will never be hit. This assignment:

```
Timeout = GLO:NumberSeconds
```

will be updated every time the date/time is updated. Placing this code:

```
0{Prop:Timer} = 0
```

in `ThisWindow.Init`, `Enter` procedure scope (Priority 501) will turn off the Frame's timer. (Because this code is executed before the current procedure's window is opened, "0" still refers to the Frame's window.) In the C55 sample app, I implement turning off

the Frame timer. I do not do so in the C5 app; try inserting the "Display the date" extension there and you will find that the browse with the "IDLE" code never times out.

Multiple Timers

I started down this road because I needed multiple timers running simultaneously. The fact that IDLE wasn't working anyway really wasn't all that important. (Right.) Implementing multiple timers now becomes a matter of declaring another variable and duplicating two small code segments (to re-initialize the additional variable(s) in TakeEvent and to do whatever needs to be done in Event Timer).

Summary

IDLE is broken but that doesn't mean that I lose the ability to set up programmatic action after a period of inactivity. I've described three different ways to implement IDLE functionality, though the last clearly seems both the easiest to implement (least amount of typing and easiest to read) and most effective (works intra- and inter-day). If a one second timer is unacceptable, for whatever reason, one of the other techniques will serve quite well. None, however, seem appropriate outside of a single procedure. Oh, yes, a template is also possible. One (crude but effective) is included in the downloadable code.

[Download the source](#)

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

Reader Comments

[Add a comment](#)

Hi, The first instance of StarDate that I...

Good example of where Equates should be used. Miss a zero...

Another good date format is this DateTime format that sorts...

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

The Clarion Advisor: API Tricks

by Pierre Tremblay

Published 2001-05-03

Often, I need to use Windows API calls which involve passing a CSTRING, and sometimes those API calls require a NULL instead of a string. A lot of programmers will prototype the API call with a LONG in order to support a call with a NULL (set the LONG to 0). The problem is you then always need to pass the ADDRESS() of the variable instead of the variable itself. When I face a situation like this I simply define two prototypes, one with a LONG and one with a *CSTRING, and I let the compiler sort out which one I'm calling.

For example, I sometimes need to break a string into tokens, discarding a specific delimiter (such as , ; . etc.). I use the API _strtok function for this. The first call accepts the string to be parsed and a string containing a list of delimiters. Subsequent calls require a NULL as the first parameter, so _strtok knows to return the next token. When _strtok returns a null, there are no more tokens. I prototype the function this way:

```
module('Lib')
  StrTok(*cstring pCString, *cstring pDelim), ←
  cstring, raw, name('_strtok')
  StrTok(long, *cstring pDelim), cstring, raw, ←
  name('_strtok')
end
```

Here's some code that demonstrates the use of _strtok:

```
ACstring      CSTRING(60)
TokenQ       QUEUE,PRE()
AToken       STRING(10)
              END
csDelim      CSTRING(' ,;:')
AToken       CSTRING(20)

CODE
ACstring = 'This is a test string'
AToken = StrTok(ACstring, csDelim)
LOOP WHILE AToken
  TokenQ.AToken = AToken
```

[Search](#)[Home](#)[COL Archives](#)[Subscribe](#)[New Subs](#)[Renewals](#)[Info](#)[Log In](#)[FAQ](#)[Privacy Policy](#)[Contact Us](#)[Downloads](#)[PDFs](#)[Freebies](#)[Open Source](#)[Site Index](#)[Call for](#)[Articles](#)

```
ADD(TokenQ)  
  AToken = StrTok(0, csDelim)  
END
```

You might use `_strtok` to parse a string from an INI file or a table record; each token could represent a valid choice for an entry field, for instance.

This approach isn't limited only to `CSTRING`s. You can use it for any structure (i.e. a typed group) where the API call allows you to pass a `NULL` instead of the structure itself.

[Download the example application](#)

Pierre Tremblay has worked in the programming and corporate world for the last 16 years, and has been as an independent contractor for TopSpeed Consulting Division since April 1998. He is also a member of Team TopSpeed.

Reader Comments

[Add a comment](#)

**What about just making the parameter ommittable? I seem to...
another approach is to make a generic NullStr variable....
< *Cstring> does do what you want per Alexey the below...
Just to clarify ; The strtok() function is not a windows...
I wasnt aware that omitable parameter will default to a...**

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca

Reborn Free**CLARION**
*online*published by
CoveComm Inc.

Clarion MAGAZINE

Introduction to SQL - Part 4

by **Dave Harms**

Published 2001-05-01

April was a busy month, so I'm a little behind on this [introduction to SQL series](#). I've received several questions about the articles, mainly related to the overall concepts of SQL development. These questions suggest that there's still some confusion among Clarion developers over what SQL is all about.

I've been asked on a few occasions just how SQL development differs from non-SQL (typically TPS) development. The answer to this question is "it depends." You can choose a development style anywhere on the continuum from "almost identical to TPS" to "radically different from TPS."

I don't care if it's SQL

It's quite possible to create a Clarion application that can run on either a TPS database or a SQL database, and the only thing you have to change is the driver. You will need to stick with data types common to all the drivers you plan to use, but other than that you don't need to make any special accommodation. You don't need to think about your development in a different way, except to the extent that you need to learn how to create or maintain a SQL database. And you need to make sure that each of your tables has a [primary key](#).

I suspect that a lot of Clarion developers who do SQL start off with this approach. Perhaps they're looking for better network performance, or maybe SQL is one of the client's requirements. In any case, the point is that you can treat SQL tables the same way you treat TPS tables (or files, if you prefer that terminology). In this situation your application assumes no intelligence other than its own is at work manipulating the database, and the SQL server functions simply as a repository for data. You ask for data, you get it. You update data, it's updated. You delete, it's gone. A SQL server used this way doesn't take any additional action based on what you ask it to do.

Your application will automatically take some minimal advantage

[Search](#)

[Home](#)

[COL Archives](#)

[Subscribe](#)

[New Subs](#)

[Renewals](#)

[Info](#)

[Log In](#)

[FAQ](#)

[Privacy Policy](#)

[Contact Us](#)

[Downloads](#)

[PDFs](#)

[Freebies](#)

[Open Source](#)

[Site Index](#)

[Call for](#)

[Articles](#)

of any SQL database server's special capabilities, primarily when you're dealing with a browse that uses related tables. In older versions of Clarion browses read files directly, using the file driver; in Clarion ABC all such file access is handled by a Clarion `VIEW` structure, which is a sort of logical table which can contain related tables. Here's an example of a `VIEW` structure that combines three tables using a `JOIN` to display authors and their articles:

```
BRW1::View:Browse VIEW(Names)
    PROJECT(nam:LastName)
    PROJECT(nam:FirstName)
    PROJECT(nam:NameID)
    JOIN(aat:AuthorID,nam:NameID)
        PROJECT(aat:ArticleID)
        JOIN(Art:PRIMARY,aat:ArticleID)
            PROJECT(Art:Title)
            PROJECT(Art:ArticleID)
    END
END
END
```

The primary table in this Clarion `VIEW` is the `Names` table. In the AppGen file schematic, this is the first table listed in the browse control, as shown in Figure 1. There are two additional tables in the `VIEW`: `AuthorArticle` is a linking table which manages a many-to-many relationship between `Names` and `Articles`.

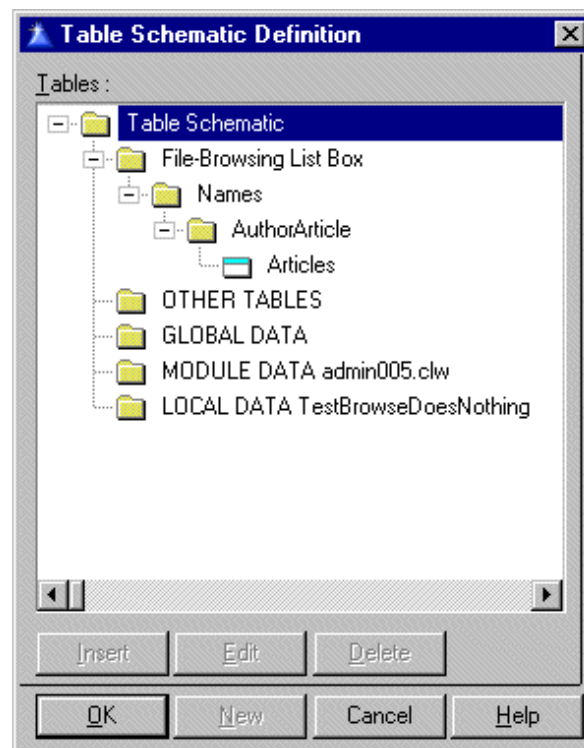


Figure 1. The browse file schematic

If you use this `VIEW` with a flat-file database, Clarion will retrieve all of fields in each table record, even though only a few of these fields are actually listed in the `VIEW`. That means you get a lot

more network traffic than you really need, and performance will suffer. If you use a SQL database, Clarion will generate a `SELECT` statement instead, and that statement will only retrieve the required fields. Here's a `SELECT` statement that corresponds to the above `VIEW` structure (I created this using the `\c55\bin\trace.exe` utility):

```
SELECT  A.NameID, A.FirstName, A.LastName,
        A.Company, A.Country, A.Email, A.UserID,
        B.AuthorArticleID, B.ArticleID,
        C.ArticleID, C.Title
FROM    Names A
        LEFT OUTER JOIN AuthorArticle B
          ON  A.NameID= B.AuthorID
        LEFT OUTER JOIN Articles C
          ON  B.ArticleID= C.ArticleID
ORDER BY B.AuthorID ASC, B.ArticleID ASC,
        C.ArticleID ASC
```

Although there are numerous fields in the `Names` table, only seven of these fields are named in the `SELECT` statement. You're probably wondering why seven, since just three are listed in the `VIEW`. As near as I can tell, ABC adds these fields automatically because they're key components. At least seven is better than 38, which is how many fields there really are in `Names`. Of course, when you bring up an update form, ABC will retrieve all of the fields in that row.

NOTE: In my tests with MySQL, ABC reports and processes, unlike browses, automatically retrieved *all* fields in the table(s), thereby removing the network performance benefit enjoyed by ABC browses running on SQL data.

In SQL, tables can be associated with a `JOIN` statement, such as this:

```
Names A LEFT OUTER JOIN AuthorArticle B
      ON A.NameID= B.AuthorID
```

There are several different kinds of joins. In a `LEFT OUTER JOIN` the SQL server will look for records for the left-side table, and find matching records on the right side table. If there are no matching records on the right side, the server supplies `NULL` values for the right side fields. This is the kind of join most Clarion programmers use, whether they realize it or not.

Notice that the `Names` table is defined in the `SELECT` statement as `Names A`, not just `Names`. The `A` is an alias for the `Names` table. Since you can have identical field names in different tables, you often need to prefix the field with the table name, as in `Names.NameID`. But that can lead to a lot of typing, so SQL allows the use of an alias. In this case, `A.NameID` is the same as `Names.NameID`. The Clarion view engine assigns these aliases

alphabetically, beginning with A.

Finally, the JOIN has to specify which are the linking fields. The Clarion view engine uses the ON syntax:

```
ON A.NameID= B.AuthorID
```

All of the above code comes from a straight ABC application that would work with SQL or TPS tables. The only difference is the file driver. So even though you don't make any special allowances for SQL, you can still get some of the speed and performance benefits of SQL.

Tuning for SQL

Although stock ABC SQL applications work, there's a whole world of functionality out there for SQL developers. Typical server features include:

- mass updates – why write a process to do something, when a single SQL will accomplish the same result?
- server-side autoincrementing of keys
- enforcing referential integrity
- stored procedures – SQL code which can be called at any time
- triggers – ability to execute a stored procedure when a particular event happens

I'll take a brief look at each of these areas, and point out some of the issues for Clarion developers.

Mass updates

Clarion developers are used to applying updates to one record at a time. With SQL, you can update large numbers of records with a single statement. For instance, let's say I've been inconsistent in storing country information in my Names table. In some cases, the country value for the United States of America is 'USA', in others 'US'. To change all instances of 'US' to 'USA' I can execute the following statement using PROP:SQL:

```
UPDATE Names SET Country='USA' WHERE Country='US' ;
```

This kind of capability doesn't necessarily have a bearing on how you design your applications, except that you can probably dispense with some of your own client-side code. Of course, you'd never allow this kind of inconsistency to appear in your data in the first place, right?

Server-side autoincrementing

Good database design requires you to have a unique identifier for each row in a table, and in most cases you'll accomplish this using an autoincrement key. Traditionally, Clarion applications

autoincrement by retrieving the record in the table with the highest key value, incrementing that value by one, inserting a record with the new value (to reserve that auto-incremented number), and changing the current action from an insert to a change (even though the form still appears to be inserting a new record). With a SQL database, you have the option of letting the server do the auto-incrementing, which is generally faster and more reliable. But this is not as straightforward as it may seem.

When you're doing a simple insert into a table, everything is fine – you may need to supply a `NULL` value for the primary key field, but the server will take care of the rest. The difficulty arises when you try to add related (child) records using the parent's update form. If you've just inserted the parent record, you won't have a value for the primary key field. That value exists, but your form has only inserted the record; it hasn't retrieved that record to find out the field value.

There are various ways around this problem. For MS SQL Server, Jim Kane has written some code to retrieve the `@@identity` variable, which contains the value of the last autoincrement identifier for the current connection. This way you can assign the correct parent id to the child record.

SoftVelocity is working on templates that are designed to work specifically with SQL databases, and presumably deal more elegantly and directly with the SQL back end on autoincrementing and other issues. For more information on the SQL templates contact sqldev@softvelocity.com.

Enforcing referential integrity

I can't claim much experience with server-side relational integrity (RI), because most of my SQL work is with MySQL, which doesn't provide this capability. Most databases do let you set various update and delete check constraints in much the same way as you're probably accustomed to setting these constraints in the Clarion dictionary editor. If you decide to handle RI on the server, you should select the appropriate server side constraint in the Clarion dictionary editor, as shown in Figure 2.

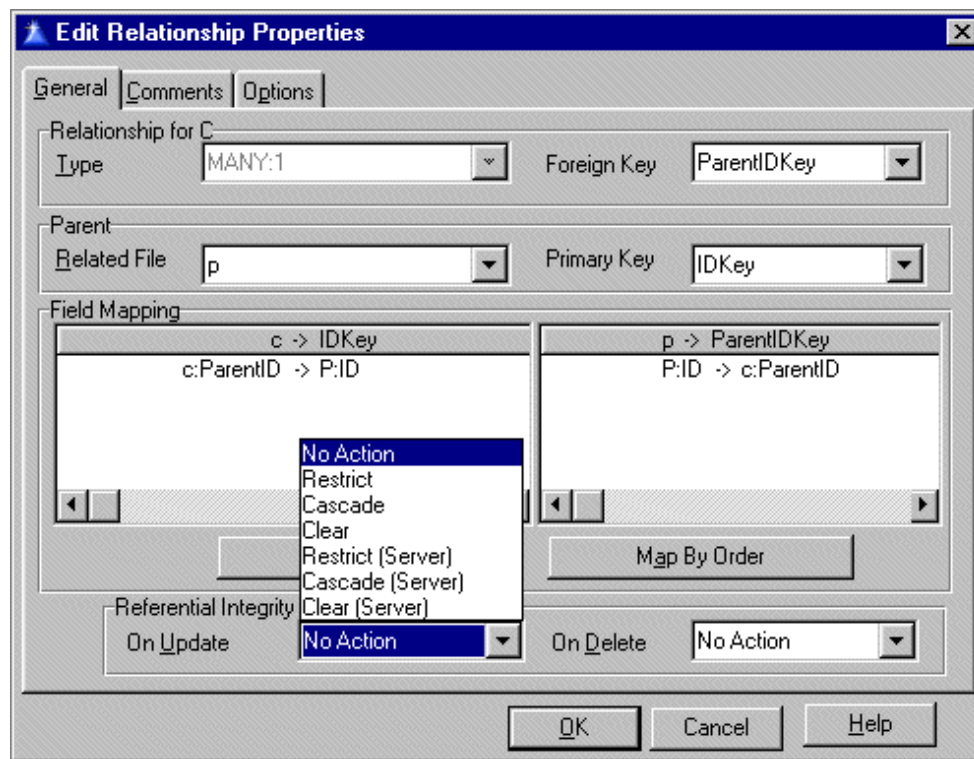


Figure 2. Choosing a server-side RI constraint

Setting server-side constraints in the dictionary doesn't create any server-side code; this is just a way of documenting that the server will handle the RI issues.

Stored procedures and triggers

Closely related to RI issues is the use of stored procedures and triggers. SQL is a query language, but it's also a data definition language, and in many ways a full-fledged programming language. With most SQL servers you can store SQL code on the server, as a procedure, and call that code from your Clarion applications with `PROP:SQL`.

A trigger is similar to an RI constraint in that a particular event (such as an update or delete) triggers an action. In fact, you can implement RI constraints as triggers, if you like. The trigger can contain all the necessary SQL code, or it can call stored procedures instead of, or in addition to, its own SQL code.

Stored procedures and triggers have the most potential to radically alter your approach to database development. Much of the code in your application represents business rules, or standard approaches to handling certain kinds of data. The more of those business rules you move to the server, the simpler the client program becomes, and the safer it is to let other applications work with the data, since the server enforces the business rules no matter which client updates the database. You can make the database itself relatively bulletproof, with enough effort.

Moving all this code to the server can present disadvantages as

well as advantages. You'll need to learn how to express your Clarion code as SQL code, and that takes some effort. If you move your application to another SQL platform, chances are you'll also have to rewrite some of your SQL code, since as Mike Gorman points out, there really is no firm and fast SQL standard. And you'll be creating a much more complex database which requires a greater level of understanding and perhaps administration.

Which way do I go?

The benefit you get from moving from a flat file database (like TPS files) to a SQL database is proportional to the degree to which you use the SQL server's capabilities. If you simply change drivers (assuming your datatypes are compatible with the SQL server), and your browses typically do not retrieve most or all of the fields in a table, then you should see better network performance. You may see better raw data access speed on the server as well, but I've never benchmarked raw TPS speed against any SQL server, so I can only guess that there will be some wide variation, depending on which SQL server you use.

Server-side processing like check constraints for referential integrity and autoincrementing further reduces network traffic, although auto-incrementing can cause some additional headaches, as I described earlier.

If you go all out and implement triggers and stored procedures, you can reduce network traffic by another notch or two. Although minimizing network traffic is an important goal for most developers, do keep in mind that you won't improve performance if the server doesn't have the processor speed and/or bandwidth to keep up with requests. You need to strike a balance between what the client computers are capable of, what the server can do, and how fast you can get data between the two. All other things being equal, however, there are significant benefits to moving business logic from the client machines to the SQL server.

Resources

Whitemarsh SQL papers	http://www.wiscorp.com/sql99.html
SQL.ORG tutorials	http://www.sql.org/online_resources.html
SoftVelocity SQL templates (under development)	sqldev@softvelocity.com
CCS SQL templates	http://www.cscowboy.com/

David Harms is an independent software developer and the editor and publisher of Clarion Magazine. He is also co-author with Ross Santos of Developing Clarion

for Windows Applications, published by SAMS (1995). His most recent book is [JSP, Servlets, and MySQL](#), published by HungryMinds Inc. (2001).

Reader Comments

[Add a comment](#)

Copyright © 1999-2001 by CoveComm Inc. All Rights Reserved. Reproduction in any form without the express written consent of CoveComm Inc., except as described in the [subscription agreement](#), is prohibited. If you find this page on a site other than www.clarionmag.com, email covecomm@mbnet.mb.ca