Clarion Magazine

Using KEYSTATE For Backdoors And Other Tricks

The Clarion KEYSTATE function returns the status of the "shift type" keys (Shift, Ctrl, Alt), the lock keys (Caps Lock, Num Lock, Scroll Lock), and the Insert key (overwrite or insert). It might sound like KEYSTATE isn't good for much more than displaying information on the status bar, but in fact this function is a great tool for detecting unusual keystroke combinations (including when the numeric keypad has been used), which you can use to implement hidden features in your applications.

Posted Friday, February 01, 2002

Eleven Winners In ClarionMag Sweeps First Draw

The first draw in the ClarionMag Sweepstakes is a done deal - 11 winners have been emailed notification of their prizes. If you missed out, don't despair. The Sweeps continues, with the grand prize of either a Compaq iPAQ or an ETC-III regisration (winner's choice) to be awarded in February.

Posted Friday, February 01, 2002

Weekly PDF for January 27 - February 2, 2002

All ClarionMag articles for January 27 - February 2, 2002 in PDF format.

Posted Monday, February 04, 2002

January 2002 PDF Now Available

The January 2002 PDF is now available for download.

Posted Monday, February 04, 2002

Managing Table Opens In ABC

Clarion applications have automatically managed opening and closing tables for years. ABC applications continue this practice. The templates, under control of global template

News

Support FTP program Using Wininet

TPS.repair Templates v1.2

gREGPlus Discount

New Coollook Template Demo

Conversion Templates Released

ClarionSearch Adds Newsgroups

Fomin Report Builder & PDF-Tools

ETC-III In Three Months

Icetips Wizards 1.05 released

INN Bio for 20-Feb-2002

Coollook Template Adds Outlook Style

SealSoft xWord Library v1.7

Gitano's gBuddy Now $49

gQ Competitive Upgrade Pricing

SysMonthCal Released

options, handle this task using the RelationManager objects. So you don't have to worry about how or when your tables are opened, right? Wrong.

Posted Monday, February 11, 2002

The Clarion Advisor: LINK Tricks

Sometimes you want to tell Clarion to include a particular library in the link process. In the template language you can do this with the #PROJECT statement. But what if you're just using straight Clarion code? Mark Goldberg shows a quick and easy way to include LIBs and other resources in your app.

Posted Wednesday, February 13, 2002

Running Totals: A Reader's Perspective

Steffen Rasmussen revisits Steve Parker's article on Running Totals, and contributes a template.

Posted Thursday, February 14, 2002

Rebasing Third Party DLLs

In the first two installments in this series Carl Barnes explained the principles of rebasing 32 bit DLLs to minimize load times and better utilize memory. This time around, Carl shows how to rebase a DLL from someone else such as a third party vendor.

Posted Friday, February 15, 2002

Weekly PDF for February 10-16, 2002

All ClarionMag articles for February 10-16, 2002 in PDF format.

Posted Monday, February 18, 2002

The CASE Statement Revisited

When you have complex program logic, nested IF statements can easily get out of hand. Brian Staff finds another way using a non-traditional CASE statement syntax.

Posted Tuesday, February 19, 2002

Please, User Dearest, Complete The Fields

In the spirit of user-friendliness, Steve Parker implements form field validation with INCOMPLETE().

Posted Wednesday, February 20, 2002

xQuickFilter v2.10

Clarion Wins In Dutch RADrace

INN Bio for 13-Feb-2002

SetupBuilder 4 Web Beta Testers Wanted

Price Reduction On SetupBuilder For Clarion Upgrade

gCal 3.0 Released

SysDTP 1.1 Released

Legacy EmailReport 2.0 Released

xTipOfDay 1.2 Released

Tiger Programs Releases Skin Add-On

New Bundles From solid.software

Spreadsheet Wizard 2 Now Supports C55 Legacy

EmailData 1.2 Available

CompactFlash Compression Template

New Addition To Free gTools

EmailReport 2.0 Template Available

CrossTab Wizard 2 Upgrade, February Discounts

Gitano Software Source Code Available

Using Real Icons In The Listbox Header - Part 1

For quite some time Steffen Rasmussen has been working with the idea of sorting listboxes by clicking on the header, and showing the sort order with an icon. His previous solution (described in Clarion Magazine) used characters in the header to show sort orders; in this series, Steffen shows how to use real icons via buttons. Part 1 of 3.

Posted Friday, February 22, 2002

EasyVersion 1.00 Released

EasyVersion 1.00 is now available. Features include: Automatic Build number generation; Automatic generation of fields with Version information for use in Splash and About screens (using Control templates); Encryption of constants containing a version information; Version Stamping of EXE and DLL with the generated version information in 32 bit applications; Documenting of process of generation of application using a LOG-file. ABC or Legacy, Clarion 5.0B or Clarion 5.5, 32 bits only to use version stamping. Demo available.

Posted Friday, February 22, 2002

Clarion Magazine Begins Fourth Year Of Publication

Hard as it may be to believe, this February marks the beginning of Clarion Magazine's fourth year of publication.

Posted Friday, February 22, 2002

Weekly PDF for February 17-23, 2002

All ClarionMag articles for February 17-23, 2002 in PDF format.

Posted Monday, February 25, 2002

Using Real Icons In The Listbox Header - Part 2

For quite some time Steffen Rasmussen has been working with the idea of sorting listboxes by clicking on the header, and showing the sort order with an icon. His previous solution (described in Clarion Magazine) used characters in the header to show sort orders; in this series, Steffen shows how to use real icons via buttons. Part 2 of 3.

Posted Tuesday, February 26, 2002

INN Bio for 06-Feb-2002

Function Points Material Posted

Taboga To Release Barcode Library 1.1

Thin Client Data Demo With Update Controls

AnalyZe.IT Download Site

AnalyZe.IT Newsgroup

EasyMultiTag 1.00 Released

Linder SetupBuilder 4.01a Available!

Oracle Dictionary Tools Released

PDF-XChange Adds PDF-Tools

The ClarionMag Sweepstakes Ends Today, Thursday Feb 28th!!

Time is running out on the ClarionMag Sweepstakes! All entries must be in by Thursday, February 28, 2002. The grand prize of either an ETC-III registration or a Compaq iPAQ (winner's choice) will be drawn March 1.

Posted Thursday, February 28, 2002

A Closer Look At Required Fields

Following reader response to his first article on required fields, Steve Parker takes a second look at this tricky subject.

Posted Thursday, February 28, 2002

Using KEYSTATE For Backdoors And Other Tricks

by Carl Barnes

Published 2002-02-01

The Clarion KEYSTATE function returns the status of the "shift type" keys (Shift, Ctrl, Alt), the lock keys (Caps Lock, Num Lock, Scroll Lock), and the Insert key (overwrite or insert). These keys are different from the rest of the keys on the keyboard in that they don't return a KEYCODE value to Clarion when pressed. It might sound like KEYSTATE isn't good for much more than displaying information on the status bar, but in fact this function is a great tool for detecting unusual keystroke combinations (including when the numeric keypad has been used), which you can use to implement hidden features in your applications. In this article I'll walk you through detecting key states, and I'll give some examples of useful hidden behaviors.

KEYSTATE returns a bitmap in a LONG with one bit turned on for each of the keys it covers. The Help file has a table a table showing all of these bit values, but the equates for these values are *not* included in the standard Clarion equates. If you're going to use KEYSTATE frequently it's best to add the equates I have defined below to the EQUATES.CLW file. (Keep in mind that you'll need to propagate these changes to future releases of Clarion, unless SoftVelocity decides to include these definitions in EQUATES.CLW.)

```
KeyState:Shift      EQUATE(0100h) !Shift key is pressed
KeyState:Ctrl       EQUATE(0200h) !Ctrl key is pressed
KeyState:Alt        EQUATE(0400h) !Alt key is pressed
KeyState:EnterOnNum EQUATE(0800h) !Num Pad Enter Key,
                                  ! n/a with MESSAGE()
KeyState:CapsLock   EQUATE(1000h) !Caps Lock is On
KeyState:NumLock    EQUATE(2000h) !Num Lock is On
KeyState:ScrollLock EQUATE(4000h) !Scroll Lock is On
KeyState:InsertKey  EQUATE(8000h) !Insert Mode is On,
                                  ! Off is Overwrite
```

Most of these equates have obvious meanings. The Shift, Ctrl and Alt bits return true if the user currently has those keys held down. The appropriate lock bits are set to true if the Caps, Num or Scroll Lock key is toggled On (the keyboard light is lit). The Insert Key (8000h) returns true if in "Insert Mode" and false if in "Overwrite Mode".

The one odd item is named "extended" in the help; it has an equate value of 0800h. This bit is on if the Enter key on the number pad was pressed. In my testing this only works in an Accept loop and not with MESSAGE . Also it does not work with a menu ITEM. Since I find KEYSTATE most useful with MESSAGE I do not use this bit, but one possible use for it (with example code in the help) is to change the enter key from the numeric pad into a tab key. This might be useful on a window that requires a lot of numeric entry.

MESSAGE back doors

Frequently I'll display a message telling a user he cannot proceed. But for testing or tech support I want to be able to get by these messages. Conveniently the MESSAGE function does not care if the user is holding down the Shift, Ctrl or Alt keys; it still returns its normal Button: value. I can then use the KEYSTATE function to see if a special combination of keys were down when the button was pressed. In the below example I want to prevent the user from overwriting a file, but want to be able to bypass the restriction if I'm holding the Ctrl+Shift key as I click OK:

```
LOOP WHILE EXISTS('Payables.sem')
   MESSAGE('You must process Payables first!')
   IF ~BAND(KEYSTATE(), Keystate:Ctrl+Keystate:Shift)
      RETURN(0)
   END
   REMOVE('Payables.sem')
END
```

For anything important you always want at least two keys involved. My main reason is that a Shift or Ctrl key can get stuck down and then suddenly your user is opening many backdoors. The code above is also simple to crack because simply pressing all the keys will open the backdoor. A more secure method would be to check the entire key state for a specific combination. For example, the code:

```
IF KEYSTATE()=Keystate:Ctrl+Keystate:Shift
```

requires the user to have the Ctrl and Shift down with all locks off. One snag with this is the Insert mode must be off, and there is no keyboard light to tell the status

of Insert. You could strip off keys you do not care about (e.g. Insert and NumLock) with the code

```
IF BAND(KEYSTATE(),0FFFFh - KeyState:InsertKey - |
  KeyState:NumLock ) = Keystate:Ctrl+Keystate:Shift
```

Another way is to BAND and explicitly keep what you want to consider, then compare to what you pressed. For instance to require the user to have Ctrl and Shift pressed, but not Alt or Scroll Lock, you would use the following code:

```
IF BAND(KEYSTATE(), KeyState:Shift + KeyState:Ctrl |
  + KeyState:Alt + KeyState:ScrollLock) |
  = KeyState:Shift + KeyState:Ctrl
    MESSAGE('You pressed Ctrl+Shift but not Alt or ScrollLock')
END
```

Clarion windows and controls

As in the MESSAGE example above, you can use KEYSTATE in the ACCEPT loop with any Clarion window, menu, toolbar or control. For example, in the Accepted event of a menu item you could check for a specific key state and bypass a limit on only one copy of a given procedure active at one time.

You can also bypass or enable print preview on a report this way. In my applications, invoices always print to the printer and are never previewed. But for my own testing I want to be able to preview. To do this, in the AskPreview method I set the SkipPreview property. To be safe I normally implement this as a MESSAGE that asks if preview is desired. I grab the key state at the start of the procedure (ThisWindow.Run) and save it; otherwise the user must hold down the Shift key for the entire report. Here's the code:

```
! Data Embed --
   KeysDown  LONG
! ThisWindow.Run – before embed --
   KeysDown = KEYSTATE()
! ThisWindow.AskPreview – before embed --
   IF BAND(KeysDown, Keystate:Shift)
      CASE MESSAGE('Preview Invoices',, ICON:Print, |
           BUTTON:Yes+BUTTON:No, BUTTON:No)
      OF BUTTON:Yes
         SELF.SkipPreview = False
      OF BUTTON:No
         SELF.SkipPreview = True
      END !CASE
   END !IF
```

Debugging

Most of us like to debug by throwing in a few MESSAGE function calls. Having these messages in all the time can be distracting. One quick-and-dirty way to turn these on/off is to check the KEYSTATE . I like to use the Scroll Lock key for this:

```
ShowDebug = BAND(KEYSTATE(),Keystate:ScrollLock)
...
IF ShowDebug
  MESSAGE('X = ' & X )
END
KEYSTATE and loops
```

KEYSTATE and loops

The key state is only updated when the ACCEPT loop cycles or a MESSAGE is displayed. This is the case for all Clarion keyboard related functions (e.g. KEYCODE, KEYBOARD). You cannot use KEYSTATE to exit a tight loop unless it's an ACCEPT loop. The below code only display the second message if you press the Shift key when you click on the first MESSAGE (and then only on the first iteration, after which the loop exits).

```
MESSAGE('Press OK to start loop.' & |
     'Press Shift to stop loop','Loop Test')
LOOP Idx = 1 TO 1000000
   IF BAND(KEYSTATE(),Keystate:Shift)
      MESSAGE('Never see this after 1||Idx=' & Idx )
      BREAK
   END
END
MESSAGE('Loop Done')
```

Summary

Certain keys, including Insert, Scroll Lock, Alt and Ctrl do not generate keycodes to the Clarion Accept statement; instead, you use the KEYSTATE function to detect the use of these keys. KEYSTATE is a convenient way to implement hidden behavior, such as security overrides and toggled debug messages, in your applications. See the downloadable source for an example you can use as a starting point in your explorations.

Download the source

Carl Barnes is an independent consultant working in the Chicago area. He has been using Clarion since 1990, is a member of Team TopSpeed and a TopSpeed Certified Support Professional. He is the author of

the Clarion utilities CW Assistant and Clarion Source Search.

Reader Comments

Add a comment

Clarion Magazine

Eleven Winners In ClarionMag Sweeps First Draw

Published 2002-02-01

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

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

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

registration. See the sweeps page for details.

Reader Comments

Add a comment

Managing Table Opens In ABC

by Jim Morgan

Published 2002-02-11

Clarion applications have automatically managed opening and closing tables for years. ABC applications continue this practice. The templates, under control of global template options, handle this task using `RelationManager` objects. So you don't have to worry about how or when your tables are opened, right? Wrong.

Clarion makes management of table opens and closes easy, but it takes a very conservative approach. In ABC applications, the templates generate `Relate:`*`tablename`*`.Open` and `Relate:`*`tablename`*`.Close` (if this table is related to other tables) or `Access:`*`tablename`*`.UseTables` (if there are no related tables) for each table in your tables schematic for every procedure. This means that a form in a browse/form pair will always attempt to open a table already opened by the browse that called it.

A table open is one of the most expensive operations you can perform. Threaded tables need to be opened only once per thread. Internally Clarion maintains a use count on each table. The use count is increased with *`tablename`*`.Open` and decremented with *`tablename`*`.Close` and their related operations. Clarion only attempts to physically open a table when the use count is zero. However, the management of the use count involves some overhead. This overhead is not significant when it involves a small number of tables from a typical user event. However, repeated table opens inside a looping process make for extremely poor performance. This means that if you write intensive processes to process data using structured procedure and functions calls, you need to optimize the file openings for good performance.

The amount of time that is takes to execute a single `RelationManager` open call

(look for `Relate:tablename.open` in the code) can vary greatly, and is usually in the range of .01 to 30 seconds. Table opening time depends on, but is not limited to, the following factors:

- the number of related tables as defined in the dictionary.
- whether the table is already open in the thread or in the application.
- whether lazy opens are turned on or off for the application. (This is seen as 'Defer opening files until accessed' in the application's global file control extensions.)
- the table driver being used.
- whether a connection to the database is already established.

Improper table management creates GPFs, especially with SQL. Different table drivers behave differently on opens. With SQL-based tables you're not physically opening the table; instead you're asking the SQL server for the data. The driver first checks to see if there's a connection to the server, and if there isn't one, it establishes one. The first connection to the table for the user typically takes 0.1 to 0.35 seconds per table. With lazy opens turned off, 40+ tables could be opened on thread initiation without any visible display to the user, resulting in delays of over 15 seconds. Therefore, you do not want to reestablish a server connection every time a thread is launched.

Use an `Access:tablename.Open` call during startup to obtain the necessary information like user rights and validate connections instead of a `Relate:tablename.Open`. This allows the absolute minimum number of tables to be open and the program loads quickly. Once the user signs in and the menu is displayed, open the essential tables that are always accessed in the Frame's thread. This allows subsequent threads to start up quickly without reestablishing the connection.

Different table drivers have different tolerance for poorly managed table opens. Topspeed tables can be opened once and closed a hundred times without problem. Topspeed tables can also be opened on a thread and not closed. If the thread terminates and the table is reopened, that's no problem. With SQL tables, you will generally get a GPF in these situations. The underlying classes don't protect the application programmer with a complete termination of the thread, so you have to be careful with the code you write. This means that if you get sloppy and forget to close a SQL table and terminate a thread, the next time you start the same thread ID and open the same table you will GPF.

> **Note**: At Mitten Software, we turn lazy opens off because of significant problems that it created in earlier versions of Clarion. I would love to turn

it on. I have not tested this problem with C5.5g. However, I have spoken to others who say the problems still exist, especially with SQL. If you are having success with lazy opens, please post a reader comment below.

Reviewing the table open management can be helpful when you are tuning an application or solving GPFs in SQL. There are four different types of information you should track:

1. How many times was the table opened in total?
2. Does the table show no use when the thread is closed?
3. What is the use count of the related tables?
4. What procedures were executed, and in what order?

Supplementing the base classes

You can track this information by changing the Clarion base classes as we have done at Mitten. This will create future maintenance issues since your changes need to be reintroduced when the class is updated by Clarion. The changes to ABFile.clw are highlighted below. ABFile.clw has the source code to the `FileManager` and `RelationManager` classes. The `FileManager` class has most of the single table methods. The `RelationManager` class has most of the table methods involving related tables. All of the changes have a signature of the developer's initials and the date of the change. If you follow this practice you can compare the source code and quickly apply any future updates.

```
MEMBER
INCLUDE('TraceOpt.clw') !JM 6/21/01
```

The include statement controls the behavior of the table debug settings. When you want to enable the debugging, just place a TraceOpt.clw file with TraceFiles, TraceCloses, and TraceProcs switch settings in the application directory. The default TraceOpt.clw without debug is stored in the `\Libsrc` folder. The redirection file will look at the current folder before `\libsrc` for CLW files. The file contains the three trace switches and a name for the open/close trace file, as follows:

```
!Traceopt.clw
TraceFiles  EQUATE(0)      !Standard Clarion switch,
                           ! non-zero to trace tables
TraceCloses EQUATE(0)      !Non-zero adds table opens
                           ! information to the debug table
TraceProcs  EQUATE(0)      !A non-zero adds a procedure
                           ! trace to the debug table.
e_iniName   Equate('c:\temp\fileclos.txt')
```

An additional variable is conditionally added to the `FileThreadQueue` in ABTable.clw, but with the compile directive to eliminate all overhead in the final release.

```
FileThreadQueue     QUEUE,TYPE      ! QUEUE of status of
                                    !  all table buffers
Id                    SIGNED        ! Thread number
Used                  BYTE          ! Set True when table is
                                    !  actually opened
Opened                USHORT        ! Table opened counter
  COMPILE('xxx',TraceCloses)        !jm 6/21/01
HardOpened            USHORT        ! This Table opened
                                    !  counter !jm 6/21/01
  xxx                               !jm 6/21/01
AtEOF                 BYTE          ! End of Table flag
AutoIncDone           BYTE          ! Auto-increment done or not flag
LastError             USHORT        ! Last error identifier
                    END
```

The `FileManager.Construct` method deletes the debug file every time the program loads. Treat the debug file as an INI file. This is easy to program and doesn't hurt performance too much, but you must be careful: if the debug file gets over 32K in size, the file will not be properly maintained.

```
FilesManager.Construct PROCEDURE
  CODE
  COMPILE('xxx',TraceCloses)
  Remove(e_IniName)
  xxx
```

Next, change the `FileManager.Close` method to log soft and hard use counts.

```
FileManager.Close PROCEDURE
  CODE
    SELF.InClose += 1
    SELF.SetThread
    FilesManager.NoteClose(SELF)
    IF SELF.Info.Opened
      SELF.Info.Opened -= 1
      IF ~SELF.Info.Opened
        CLOSE(SELF.File)
        SELF.Info.Used=False
      END
      PUT(SELF.Info)
?     ASSERT(~ERRORCODE(),↵
        'Unable to store thread specific file information.')
  COMPILE('xxx',TraceCloses)        !jm 6/21/01
      If SELF.Info.Opened
        PUTINI('SoftCount', Thread() & NAME(SELF.File), ↵
          SELF.Info.Opened , e_IniName)
      Else
```

```
        PUTINI('SoftCount', Thread() & NAME(SELF.File), , ↵
          e_IniName)
      End
    Else
      PUTINI('SoftCount', Thread() & NAME(SELF.File), ↵
        -1, e_IniName)
  xxx                               !jm 6/21/01
    END
  COMPILE('xxx',TraceCloses)        !jm 6/21/01
    If SELF.Info.HardOpened
      PUTINI('HardCount', Thread() & NAME(SELF.File), ↵
        SELF.Info.HardOpened, e_IniName)
    Else
      PUTINI('HardCount', Thread() & NAME(SELF.File), ↵
        , e_IniName)
    End
  xxx                               !jm 6/21/01
    SELF.InClose -= 1
    RETURN Level:Benign
```

The FileManager.OpenServer handles the actual table opening; tweak it to log use counts:

```
FileManager.OpenServer PROCEDURE(BYTE HandleError,↵
        BYTE ForceOpen)
RVal BYTE,AUTO
  CODE
    SELF.SetThread
    FilesManager.NoteOpen(SELF)
    COMPILE('***',Traces)
    IF TraceFiles
      FileTablesManager.Trace('Open'&CHOOSE(HandleError=1,↵
        '(Errors):',':')&SELF.GetName())
    END
    ***
    SELF.BindFields
    IF ForceOpen OR ~SELF.LazyOpen AND ~SELF.Info.Opened
      RVal = SELF.OpenFile(HandleError)
      IF RVal
        RETURN RVal
      END
    ELSIF SELF.Info.Opened
              COMPILE('xxx',TraceCloses)    !jm 6/21/01
                PUTINI('HardOpen', Thread() ,↵
                   SELF.Info.Opened, e_IniName)
              xxx                               !jm 6/21/01
    END
    SELF.Info.Opened += 1
    PUT(SELF.Info)
?   ASSERT(~ERRORCODE(),↵
        'Unable to store thread specific file information.')
          COMPILE('xxx',TraceCloses)        !jm 6/21/01
    If SELF.Info.HardOpened
      PUTINI('HardCount', Thread() & NAME(SELF.File), ↵
```

```
        SELF.Info.HardOpened, e_IniName)
    Else
      PUTINI('HardCount', Thread() & NAME(SELF.File), ↵
        , e_IniName)
    End
    If SELF.Info.Opened
      PUTINI('SoftCount', Thread() & NAME(SELF.File), ↵
        SELF.Info.Opened , e_IniName)
    Else
      PUTINI('SoftCount', Thread() & NAME(SELF.File), ↵
        , e_IniName)
    End
          xxx                                !jm 6/21/01
    RETURN Level:Benign
```

The RelationManager.OpenCloseServer procedure is tweaked to track times opened here and maintain the use count.

```
RelationManager.OpenCloseServer PROCEDURE(↵
        BYTE Cascading,BYTE Opening)
I   BYTE(1)
Res BYTE,AUTO
   COMPILE('xxx',TraceCloses)        !jm 6/21/01
MyCount     Long                 !jm 6/21/01
            Xxx                !jm 6/21/01
  CODE
    IF Cascading
      IF SELF.LastTouched = Epoc
        RETURN  Level:Benign
      END
    ELSE
      Epoc += 1
    END
    SELF.LastTouched = Epoc
?    ASSERT(NOT SELF.Relations &= NULL,↵
        'Relation manager incorrectly initialized.')
    IF Opening
      Res = SELF.Me.Open()
      COMPILE('xxx',TraceCloses)      !jm 6/21/01
      IF Not Cascading
        Self.Me.SetThread
        Self.Me.Info.HardOpened += 1  ! File opened counter  !jm
        PUT(SELF.Me.Info)
        MyCount = GETINI('TimesOpened', Thread() ↵
          & NAME(SELF.Me.File), 0 , e_IniName) + 1
        PUTINI('TimesOpened', Thread() & NAME(SELF.Me.File)↵
          , MyCount, e_IniName)
      End
      Xxx                         !jm 6/21/01
      IF ~Cascading THEN Res=SELF.Me.UseFile().
    ELSE
      Res = SELF.Me.Close()
      COMPILE('xxx',TraceCloses)      !jm 6/21/01
      IF Not Cascading
```

```
        Self.Me.SetThread
        Self.Me.Info.HardOpened -= 1  !File opened counter !jm
        PUT(SELF.Me.Info)
      End
      Xxx                             !jm 6/21/01
    END
    LOOP UNTIL Res
      GET(SELF.Relations,I)
      IF ERRORCODE()
        BREAK
      END
      IF Opening
        Res = SELF.Relations.File.Open(1) ! Use ↵
        'public' interface to pick up VIRTUAL ness
      ELSE
        Res = SELF.Relations.File.Close(1)
      END
      I += 1
    END
    RETURN Res
```

You'll need to make some changes ABError.clw to trace the procedures. Add the include statement to check the debug equate settings and a few variables.

```
  MEMBER
  INCLUDE('ABERROR.INC'),ONCE
  INCLUDE('ABERROR.TRN'),ONCE
  INCLUDE('TraceOpt.clw')          !jm 6/21/01
  COMPILE('xxx',TraceProcs)        !jm 6/21/01

MyProcedureName    CString(51)  !Temp Storage for the last
                                !procedure recorded in ini file.
TraceNumber        Short
  xxx
```

Enhance the `SetProcedureName` to log procedures. This method is called automatically for most template-generated procedures. You can add the statement

```
GlobalErrors.SetProcedureName('ProcedureName')
```

in source procedure init and "`GlobalErrors.SetProcedureName`" in procedure kill to clear it.

```
ErrorClass.SetProcedureName PROCEDURE()
  CODE
  IF OMITTED(2)
    IF SELF.GetProcedureName()
      DELETE(SELF.ProcNames)
    END
  ELSE
    SELF.ProcNames.Name = CLIP(S)
    SELF.Procnames.Thread = THREAD()
```

```
    COMPILE('xxx',TraceProcs)      !jm 6/21/01
  IF SELF.ProcNames.Name
    MyProcedureName = GETINI('LastProc',↵
      SELF.Procnames.Thread, , e_IniName)
    IF MyProcedureName <> SELF.ProcNames.Name
      PUTINI('LastProc',SELF.Procnames.Thread,↵
        SELF.ProcNames.Name, e_IniName)
      TraceNumber = GETINI('TraceNumber',↵
        SELF.Procnames.Thread, , e_IniName) + 1
      PUTINI('TraceNumber',SELF.Procnames.Thread, ↵
        TraceNumber, e_IniName)
      PUTINI(SELF.Procnames.Thread & 'Trace',TraceNumber, ↵
        SELF.ProcNames.Name, e_IniName)
    End!IF MyProcedureName <> SELF.ProcNames.Name
  End!IF SELF.ProcNames.Name
    xxx                            !jm 6/21/01
```

Reviewing the debug file

The resulting debug file is shown below. It is divided into several sections depending on use, and within each section are variables that show the thread number and table or procedure name. [SoftCount] is the current use count and is incremented every time a table is opened directly or indirectly through a relationship. [HardCount] is the current use count showing direct opens only. [TimesOpened] counts every time a table is opened directly. [LastProc] is the last procedure to register. [TraceNumber] is used for internal counting. [nTrace] is the procedure trace by n Thread.

```
[SoftCount]
1N:\TenaSQL\SLConfig.Cfg=1   Current Use Count is 1 in Thread #1
1N:\TenaSQL\Machine.Cfg=1
1dbo.Contacts=1
1dbo.PSNote=1
2N:\TenaSQL\SLConfig.Cfg=1
2dbo.Cases=27                Current Use Count is 27 in Thread #2
2dbo.Followup=27
2dbo.Answers=27
[HardCount]
1N:\TenaSQL\Machine.Cfg=2    Hard Use Count is 2 in Thread #1
1SLMul_=1                    Hard Use Count is 1 in Thread #1
2N:\TenaSQL\SLConfig.Cfg=2
2dbo.Status=1                This file was left open in Thread #2.
[TimesOpened]
1N:\TenaSQL\SSProg_.tps=1    One physical Open in Thread #1
1N:\TenaSQL\Machine.Cfg=6
2dbo.Tables=15               15 physical Opens in Thread #2
2dbo.TokenList=28
2dbo.QNotes=35               35 physical Opens in Thread #2, wasteful!
[LastProc]
1=Main                       Main is last proc called in Thread #1
2=LoadTokenList
```

```
[TraceNumber]
1=2                          2 procedures traced in Thread #1
2=14
```

Analyzing the results

The `[softcount]` section is occasionally useful. If the numbers for a series of related tables is not the same, it indicates that an `Access:`*`tablename`*`.Close` was used instead of a `Relate:Tablename.Close`. The `[HardCount]` section should be empty upon thread closure. If any SQL table shows a use, you will get a GPF the next time that thread is started and the table is opened. The `[TimesOpened]` counts should be modest. If a table has a high count, the table should be opened at a higher level on the thread. `[LastProc]` should tell you approximately, where a GPF occurred. `[nTrace]` will show you how you got there.

Summary

Tracing your use of tables is extremely helpful for debugging and tuning performance. Using the above techniques you can make your applications and threads load faster, your processes run faster and your applications more stable.

Jim Morgan is president of Mitten Software, which has provided time-saving, value-added tools, services and accessories for the Clarion family of products for customers around the world since 1988. Mitten Software specializes in development of expert systems, accounting, order processing, distribution, manufacturing and retail systems. Mitten's products for Clarion Developers include the Boxsoft Super Templates and the latest book on Clarion, Randy Goodhew's 'Clarion Companion'. Jim is married, has one son, and enjoys skiing, golfing, boating, snowmobiling and traveling.

Reader Comments

Add a comment

Using OutputDebugString() (instead of PUTINI) would result...
For flat files you can use 'Process Explorer' from...
Jeff Slarve's Thread Manager class (available under Open...
I wonder what happens when you leave the tables open. We...
Defer Open used successfully. Hi Jim - Like you I also...
Defer Open Used successfuly - I should point out that I'm...

Clarion Magazine

Clarion News

Published 2001-11-21

Support FTP program Using Wininet

Ron Schofield has written a little FTP program called App File Transfer (AFT) so that he can give his less-than-Internet-savvy customers the ability to send him files and receive patches and updates. It's to be installed in the directory that the customers application is in. It doesn't allow for changing of directories, deletion of files, renaming of files. It also doesn't check that IE 4.01+ is installed or automatically dialup if no internet connection is found. The application doubles as an example of FTP wininet programming. If you modify or use it, please send any changes/fixes back to Ron Schofield at rschofie@hfx.andara.com.

Posted Tuesday, February 26, 2002

TPS.repair Templates v1.2

New in the TPS.repair Templates, version 1.2: support for French and Norwegian; support for UNC path names; new "fresh" mode to unconditionally apply single copy method to all data files; option to omit records that would cause dupliate key errors in single record copy method; minor bug fixes. All registered users will automatically be sent the new version free of charge via eMail. A fully functional 30 day trial version is available for download.

Posted Tuesday, February 26, 2002

gREGPlus Discount

Until Mar 2, 2002 you can save$100 on a gREGPlus full license or competitive upgrade license, or $50 on the upgrade if you are a current user. gREGPlus uses state-of-the-art encryption technology to protect your software against unauthorized use, and includes an integrated customer management and customer technical support environment that is easy to use and understand. This offer is good only when you place an order through the Gitano web site.

Posted Tuesday, February 26, 2002

New Coollook Template Demo

C&G Software has released a new demo of the Coollook Template, which adds an Outlook style menu bar to your application.

Posted Tuesday, February 26, 2002

Conversion Templates Released

Eric Jocobowitz has released Convertit, a set of English to Metric measurement conversion templates for length, volume, temperature, and more. This set of code templates is available for $29.

Posted Tuesday, February 26, 2002

ClarionSearch Adds Newsgroups

The ClarionSearch web site, by Kim Han Peck, now indexes the comp.lang.clarion, topspeed.topic.contract_jobs, and topspeed.topic.third_party newsgroups.

Posted Tuesday, February 26, 2002

Fomin Report Builder & PDF-Tools

John Verbeeten confirms that Oleg Fomin has completed testing PDF-Tools with Fomin Report Builder. All went as expected, and Fomin Report Builder generates PDF files from report output without the need for Printer drivers to be installed on an end users equipment - just add the 2 PDF-Tools Template's/DLLs to your project. Fomin joins CPCS as approved compatible with PDF-Tools.

Posted Friday, February 22, 2002

ETC-III In Three Months

It's now three months and counting to the East Tennessee Clarion Conference & Gathering. Space is still avilable, but don't wait too long.

Posted Friday, February 22, 2002

Icetips Wizards 1.05 released

Icetips Software has released Standard Wizards version 1.05. This release contains several changes and fixes needed to make the Standard edition a solid base for the Professional edition. The Icetips Wizards are ABC compatible Application, Browse, Form and Report wizards that allow the developer to customize the look and feel of their applications. Using a Wizard Editor the developer can set fonts and wallpapers for windows and reports, icons and text for buttons and all kinds of other options. This saves a lot of time when creating applications and procedure where maintaining the same look and feel is important.

Posted Friday, February 22, 2002

INN Bio for 20-Feb-2002

He's used Clarion to write software for the European Space Agency. His company recently won a Dutch software competition (see the thread titled "Clarion wins in Dutch RADrace" in the softvelocity.products.c55ee newsgroup). He's a Clarion Distributor. And he's worked as a farmhand, growing... tulips, among other things. Don't miss this week's bio about another interesting Clarionite, with more great pictures.

Posted Thursday, February 21, 2002

Coollook Template Adds Outlook Style

Coollook, C&G Software's first commercial template, lets you add an Outlook-style menubar to your applications. You populate a Coollook menubar the same way you populate a standard Clarion menubar. Works with C5.5 ABC or Legacy. Demo available. Coollook is sold by ClarionShop. Also available from C&G: Leg2ABC, a utility to help with conversion; WinUpx, a Windows end-user interface for the DOS UPX compression tool; Barcode, a template for EAN13 and UPC-A barcodes; Num2Words, a function that converts a number to words; AppRegister, a set of templates for adding a serial number to an application and aiding in registrations; TabIsEnter, to substitute the Enter key for the Tab key on forms.

Posted Wednesday, February 20, 2002

SealSoft xWord Library v1.7

SealSoft Company has released xWord Library v1.7. Changes include MergeDocument methods and use of tables. New methods include: SelectCell, SelectRow, SelectColumn, SelectTable, InsertRows, InsertRowsAbove, InsertRowsBelow, InsertColumns, InsertColumnsRight, InsertCells, DeleteRow, DeleteColumn, DeleteCells, DeleteTable, GoToTable, GoToCells, MergeOpenDataSource, MergeAddField, MergeAddFieldP; InsertDateTime. New demo and install are available.

Posted Tuesday, February 19, 2002

Gitano's gBuddy Now $49

Gitano Software's G-Buddy is a programmer's utility that helps you organize everything from simple notes to template construction and reorganization. G-Buddy includes Image, Note, Template, Message, and Color Buddy. G-Buddy can also become a toolbar. Also includes: Calendar; Calculator; Run Clipboard; Run MS Explorer; Run an app of your choice; Run G-RegDev; Minimize to toolbar and keep on top.

Posted Tuesday, February 19, 2002

gQ Competitive Upgrade Pricing

gQ from Gitano Software, Inc. is an easy-to-use querying tool for Clarion applications. Queries can be created using a template format, then sorted and filtered. Users can be up and running within minutes. Date and time fields are supported independently. Demo available.

Posted Tuesday, February 19, 2002

SysMonthCal Released

SysMonthCal is the eighth in solid.software's series of wrapper classes for the Win32 common controls. SysMonthCal wraps around the so-called "month calendar" control, which implements a calendar-like user interface. This provides the user with a very intuitive and recognizable method of entering or selecting a date. SysMonthCal doesn't use the OCX interface, but creates and handles the control via native API calls. The whole range of functionality is there, including range-selection and day states (the ability to highlight certain dates), significantly reducing the amount of code needed for implementation of the calendar. SysMonthCal comes with ABC and legacy template sets. The library is available as a DLL and a LIB version, supporting both local and external runtime libraries (32bit only). Updates and email support are free. SysMonthCal is US$39.

Posted Tuesday, February 19, 2002

xQuickFilter v2.10

SealSoft Company has released xQuickFilter v2.10. New features include: saved filters; require a filter on loading; default start filter; mouse and hot key support; new control and code templates. New demo and install available.

Posted Tuesday, February 19, 2002

Clarion Wins In Dutch RADrace

RADventure B.V the Dutch distributor for Softvelocity products has won the first prize in a RADrace organized by the publishers of Database magazine and Software Release magazine in the Netherlands. The race was held December 7 and 8, 2001, at the Advanced Development Center of Cap Gemini in Utrecht , the Netherlands. The jury consisted of well-known speakers and database authorities like Rick van der Lans, Ron Toledo, Peter Hinssen and Jan Detremmerie. The race was won using Clarion C55 Enterprise Edition patch E, and several templates and tools developed by RADventure. Magic came in a distant second. The RADventure Clarion development team took home an enormous trophy, a bottle of champagne, and this being Holland, a bouquet of flowers.

Posted Tuesday, February 19, 2002

INN Bio for 13-Feb-2002

If you like covered bridges, you will enjoy this week's INN Bio interview. If you're not familiar with covered bridges, you get a chance to see some great examples. This week's featured developer hails from the New England area of the United States. He talks about OOP and the ACCEPT statement, and a really cool calendar he's written. And we get to hear about those beautiful old bridges, too.

Posted Tuesday, February 19, 2002

SetupBuilder 4 Web Beta Testers Wanted

Linder Software is looking for beta testers for SetupBuilder 4.0. The new SetupBuilder 4 setup compiler makes it possible to generate both standard Windows and Internet-based installations (which do rely on browser plug-ins or ActiveX controls). Internal tests were successful and the SetupBuilder 4 Web Edition beta is expected next week. Beta testers must be currently registered SetupBuilder 4 Standard users!

Posted Tuesday, February 12, 2002

Price Reduction On SetupBuilder For Clarion Upgrade

Linder Software has reduced the upgrade price of SetupBuilder for Clarion to the full Linder SetupBuilder 4.01 Standard Edition from $149.00 to $129.00. To upgrade from the Clarion Edition to the full version, please use the above link. The special Web Edition offer has been extended until the end of February 2002. Users who purchased SetupBuilder Standard before 01-March-2002 are entitled to get the Web Edition for $99.00 (when available). SetupBuilder 4 Web Edition will list for $349.00.

Posted Tuesday, February 12, 2002

gCal 3.0 Released

Gitano Software has released gCal v3.0. New features include: A new drop down calendar with the the look and feel of the new calendar in Outlook XPT; A new calendar control - just point and click; A new time picker with built-in support for 5, 10, 15, 20, 30 and 60 minutes increments and various formatting options if you need them; Quick keys auto implementation. This new version is free for those that have purchased gCal since Nov 1, 2001. All other users can upgrade for a nominal fee. Source code is available.

Posted Tuesday, February 12, 2002

SysDTP 1.1 Released

Version 1.1 of SysDTP is now available from solid.software. This update is free for

all registered users. A new demo version has also been uploaded. SysDTP is a wrapper class for the "Date and time picker" common control, allowing you to use these neat little drop-down calendars in your Clarion applications.
Posted Tuesday, February 12, 2002

Legacy EmailReport 2.0 Released

Vivid Help Systems has released EmailReport (2.0) LEGACY. New features in version 2.0: Compatible with all major 3d party report templates: CPCS, RPM, DAS previewer, and Tintools previewer; Supports CC and BCC email addressing; Allows override global image shrinking factor locally for each report; Network friendly - many network users now can run it simultaneously. Current users can get a free upgrade from Clarion Shop.
Posted Tuesday, February 12, 2002

xTipOfDay 1.2 Released

SealSoft has released xTipOfDay v1.2. New features include: simplified registration input; bug fix related to multiple access. New demo and install available.
Posted Tuesday, February 12, 2002

Tiger Programs Releases Skin Add-On

Tiger Programs has released Tiger Custom Skin, a utility that lets the user modify screen properties at runtime. Properties include colors, fonts, position, wallpaper, etc. Tiger Custom Skin is $49 and includes one template, two DLLs, and two LIBs. The template lets you specify what controls the end user can modify. Designed for Clarion 5.5, 32 bits.
Posted Tuesday, February 12, 2002

New Bundles From solid.software

New from solid.software: SysBarPack, SysEntryPack and SysImagePack. SysBarPack is a bundle of bar-like controls (SysTrack and SysProgress), for $69; SysEntryPack is a bundle of entry controls (SysDTP, SysHotKey and SysIP) for $99, and SysImagePack is a bundle of our image-related controls (SysAni and SysList)for $119.
Posted Tuesday, February 12, 2002

Spreadsheet Wizard 2 Now Supports C55 Legacy

Spreadsheet Wizard 2, released in January 2001 to seamlessly integrate with Microsoft Excel™, now supports both C55 ABC and Legacy development environments. Current SW2 customers can download an update that includes the Legacy template files free of charge. February Special of 20% off applies to both the

purchase or upgrade of Spreadsheet and CrossTab Wizards. Discount do not apply to Source Code products.

Posted Monday, February 11, 2002

EmailData 1.2 Available

Vivid Help Systems has released EmailData 1.2 for C5, C55, ABC and Legacy. New features in version 1.2: Automatically embeds local images; Supports CC and BCC email addressing; Network friendly - many users can use simultaneously. Current users can get a free upgrade from Clarion Shop.

Posted Friday, February 08, 2002

CompactFlash Compression Template

Sterling Software has released CompactFlash, a template that provides high performance royalty-free compression for Clarion apps. CompactFlash works as a template wrapper forthe ZLIB DLL which is a freeware compression library written by Jean-Loup Gailly and Mark Adler. All source code is available and the compress/uncompress options are available as simple function calls. Features include: Compress any size file; Animated icon display while processing; Legacy and ABC versions; Compatible with all versions of Clarion from CW2 to C5.5; All source code supplied; No runtime royalties. Demo available. CompactFlash is $29 during the launch period.

Posted Friday, February 08, 2002

New Addition To Free gTools

Gitano Software's free gTools package now contains the gMsg Message Builder and an updated Notes application, gNts. These utilities are free to licensed Clarion users; Clarion must be installed in your system in order gTools to run.

Posted Thursday, February 07, 2002

EmailReport 2.0 Template Available

The "try before you buy" version of EmailReport 2.0 by Vivid Help Systems is now available. New features in version 2.0 include: Compatible with all major third party report templates: CPCS, RPM, DAS previewer, and Tintools previewer; Supports CC and BCC email addressing; Allows override global image shrinking factor locally for each report; Network friendly - many network users now can run it simultaneously.

Posted Thursday, February 07, 2002

CrossTab Wizard 2 Upgrade, February Discounts

Nice Touch Solutions has released an upgrade to CrossTab Wizard. By popular demand Nice Touch has moved away from the FormulaOne ActiveX control and now

uses Excel. This brings CrossTab Wizard in line with the Spreadsheet Wizard product in terms of Excel integration. Demo available. To launch this new release Nice Touch is offering a 20% discount on CrossTab Wizard purchases and upgrades during the month of February. CrossTab Wizard 1.0 to 2.0 Upgrade $63 (regular price $79); CrossTab Wizard 2.0 Purchase $135 (regular price $169). During February you can also get a special package price for Spreadsheet Wizard 2 and CrossTab Wizard 2: both products for $239. Upgrade your existing version 1.x package for $126.
Posted Thursday, February 07, 2002

Gitano Software Source Code Available

Source code is now available for Gitano Software's gCal and gCalc. Those that have the DLL version can upgrade to the source code version by paying the difference between DLL and source code prices (plus an upgrade price if applicable).
Posted Thursday, February 07, 2002

INN Bio for 06-Feb-2002

This week, the Icetips News Network is pleased to present an interview with one of Clarion's better-known developers, a programmer from Germany who has created quite a splash with his flagship product. His main competitor was so impressed (and worried) that they made a serious take-over offer. Read about that, his views on being self-employed, and an entrancing description of the area in which he lives.
Posted Thursday, February 07, 2002

Function Points Material Posted

Mike Gorman has posted a zip of function point analysis information. The zip contains background material, cost tables, a counting example for Clarion, and a summary of the counts for the database application that Mike built.
Posted Thursday, February 07, 2002

Taboga To Release Barcode Library 1.1

Taboga Software has added support for the Legacy templates to Barcode Library 1.1. Due to various reasons, with this release, the barcode library will again be a commercial product selling for $79.00 including all source code. The library will be released sometime before the end of February. Until then the freeware version will continue to be available for download. After version's 1.1 release, the freeware version will NO longer be available.
Posted Monday, February 04, 2002

Thin Client Data Demo With Update Controls

The Clarion Handy Tools recently introduced its thin client technology. A new (960K)

download is now available, with update controls. This thin client application connects to the CHT browser server and displays data using a standard browse. Both the client application and the server application are written in standard Clarion using no low level libraries except those provided by MS Windows 98 and up, plus The Clarion Handy Tools.

Posted Monday, February 04, 2002

AnalyZe.IT Download Site

AnalyZe.IT now has a U.S.A mirror download site with a 3GB bandwidth.

Posted Monday, February 04, 2002

AnalyZe.IT Newsgroup

AnalyZe.IT it now has a newsgroup for user support. There may be some Norwegian postings - don't let that stop you.

Posted Monday, February 04, 2002

EasyMultiTag 1.00 Released

New from IngasoftPlus, EasyMultiTag expands the functionality of the ABC browse. Features include: Windows-style tagging of any records (both separate and ranges); high-speed filtration and calculation of totals; instantaneous inversion; Transfer of tag information to reports and processes; Built-in support for drag & drop, EIP and QBE. Full documentation, annotated example. C5b and C5.5, ABC, 32-bits.

Posted Monday, February 04, 2002

Linder SetupBuilder 4.01a Available!

Linder SetupBuilder 4.01a is available now. This release contains several bug fixes and enhancements. You can download a small patch to update your previous version 4.0 to the latest release or you can download the full install image. SetupBuilder 4.01 costs $199.00 for a royalty-free usage license. A trial version is available.

Posted Monday, February 04, 2002

Oracle Dictionary Tools Released

RADventure Products has released the Oracle Dictionary tools template, a utility template that generates a number of Oracle scripts from your Clarion data dictionary. Some example scripts are table/index/constraints scripts, comments and dictionary compare, trigger skeleton scripts and many others.

Posted Monday, February 04, 2002

PDF-XChange Adds PDF-Tools

New from Tracker Software Products, PDF-Tools is an entirely new set of tools and functions which extend the functionality of the PDF-XChange product line. They will for a short time be offered free to all existing PDF-XChange Owners, and additionally sold as separate tool for US$449.00. PDF-Tools is entirely DLL based and does not require the installation of Print drivers. Features include: Clarion report capture and output to PDF; Create PDF bookmarks and thumbnails for easynavigation; Convert image files to PDF (BMP, GIF, JPEG, TIFF, WMF); Merge and extract pages from PDF files to create new files; Retrieve PDF file properties and where appropriate re-write; Page count, subject, author, title, etc. CPCS tested.

Posted Monday, February 04, 2002

Reader Comments

Add a comment

The Clarion Advisor: LINK Tricks

by Mark Goldberg

Published 2002-02-13

One of the things I wish Clarion had was a way to tell the linker, in Clarion code, to include a library in the project. If you're writing templates you can do this with the #PROJECT statement. But what if you're just using straight Clarion code? There is a way, using the LINK attribute for classes. Here's an "empty" class definition that demonstrates what I mean.

```
LinkMyStuff_Lib CLASS,TYPE,LINK('mystuff.lib') END
```

This class has only one purpose: to tell the compiler to include mystuff.lib as a resource during the linking phase. It will save you that little bit of effort including the LIB file in the project manually. You can also use this trick to include other resources such as icons:

```
LinkMyIcon_Ico CLASS,TYPE,LINK('MyIcon.ico') END
```

If you think of another use for the LINK attribute, post it as a reader comment below.

Mark Goldberg has been a full time Clarion programmer since CPD 2.003. He is an independent consultant living and working in Wisconsin and Florida. In recent years his primary client has been Tradesmen's Software (www.tradesmens.com) where he has integrated TGS's (www.tgs.com) VRML superset for 3D support and Wintab (www.pointing.com) for digitizer support. Mark's wife Michele Hugo does the tech writing / help file creation and is Monolith's business manager.

Reader Comments

Add a comment

Might also be a good way to conditionally link in different...

Running Totals: A Reader's Perspective

by Steffen Rasmussen

Published 2002-02-14

Dear Mr. Parker,

At the office quite few of my users have daily telephone contact with customers who want to know their account balance. This is information that is easy to look up, but if the customer disagrees, a more thorough examination of the account is required. Although the users can sort and filter the different columns in nearly all possible ways as well as make different query requests, it doesn't seem to be enough. Instead they print the balance out on paper and start calculating, marking and taking notes. In most cases they just find out that the balance is correct. So much for the paperless office.

I recently came across your article Running Totals, Mr. Parker. Although I have seen this kind of solution on my bank statements I hadn't previously thought about adding it to my application. Unlike you I store my debit and credit amounts separately. Also I don't store the balance in an external file, so a continually updated balance is not needed. Luckily for me, this removes a lot of the complexity you introduced. Now it is just a matter of removing the unused code from your Running Totals example and creating a reusable template that will do the work for the future.

You point out several problems among which at least one was not solved: "When changing tabs, calculations of the balance will be seriously impacted." The reason it wasn't solved, I guess, was probably because, as you mention: "The great secret of a positional balance is that only one sort order makes sense with this type of balance: date order." This could be, and an accountant would probably agree. But I am not an accountant. As I see it, a running balance can also be used to interpret

different compositions of the records, when filtered or sorted. In this way I hope to eliminate the users' extensive use of paper and time when they are in contact with the customers.

To provide this functionality I want to reset the queue so it always starts its calculation from the top of the record list when changing tabs. I just have to reset the queue in `ResetFromView`, before Parent Call:

```
BRW1.ResetQueue(Reset:Queue)
```

An error that wasn't mentioned in your article is that the running total is never reset when the tabs are changed. As a consequence the running total just keeps "running". To prevent this I need to reset `R_Balance` in the embedded point `ResetQueue` before the parent call, instead of resetting it in the `ResetFromAsk`. Resetting the queue is still needed.

You also mention, "if a new record is added so that it falls between existing records, the calculation will be erroneous". Your solution to reset the `R_Balance` and the queue doesn't work – at least not for me. Instead I use the embedded point `ResetFromFile` before the parent call to reset the queue.

I include with this letter a small template for running totals, as a way for me to say thank you, Mr. Parker, for all the great articles you have written, and I look forward to your article number 1000.

Although the included template does need the debit and credit fields to be separated I hope you can use it.

Yours Sincerely

Steffen S. Rasmussen

Download the source

Steffen S. Rasmussen has graduated in Computer Science from Copenhagen Business College. Since then he has worked as a programmer, system technician and network administrator, and is currently IT manager. Clarion is a quite a new language to Steffen since his only been working with it since January 2000. But what better way to learn it than by trying to teach others! Steffen has also set up a web site to collect as many examples of different user interfaces as possible to inspire Clarion developers.

Reader Comments

Add a comment

Excellent. Thank you.

Clarion Magazine

Rebasing Third Party DLLs

by Carl Barnes

Published 2002-02-15

In the first two installments in this series I explained the principles of rebasing 32 bit DLLs to minimize load times and better utilize memory. I showed how you can easily rebase DLLs you create with Clarion by adding a single line of source code to your project EXP file, or by using a template I wrote. But what happens when you want to rebase a DLL from someone else, perhaps a third party vendor?

When you rebase your own DLLs, the linker uses the `IMAGE_BASE` value you specify to write the DLL. Rebasing an existing DLL means changing binaries; luckily Microsoft has created a utility for this purpose. The utility has the easy to remember name REBASE.EXE and is part of the free MS Platform SDK.

Rebase.exe Syntax

Rebase.exe is a command line utility with no user interface. To get help on using Rebase, type `Rebase /?` at the command prompt. There is also help on MSDN if you search for "Rebase.exe". I've found the help to be lacking, and there are no less than 18 command line switches. Fortunately only one switch is required: the `-b` to specify the base address. The `-v` switch which generates verbose output is also useful. The basic syntax for Rebase.exe is:

```
rebase -b base_address -v dllname.dll [dllname.dll...]
```

The first DLL will be rebased using the `-b` specified base address. Additional DLLs are given the next free address available after the end of the previous DLL. There is no padding left between DLLs. You may list as many DLLs on the command line as will fit. You cannot use wild cards in the list.

In the previous article I discussed how to pick a base address. I would place other peoples DLLs in an address range different from the address range used by DLLs I have created. For example, my own DLLs are based starting at 0600,0000h, and I put all third-party DLLs starting at 0700,0000h. This makes it obvious that a GPF is not in *my* code.

To rebase, for example, the five Report & Presentation Manager (http://www.cwaddons.com/products/products.html) DLLs to an address of 0700,0000h I created the following batch file:

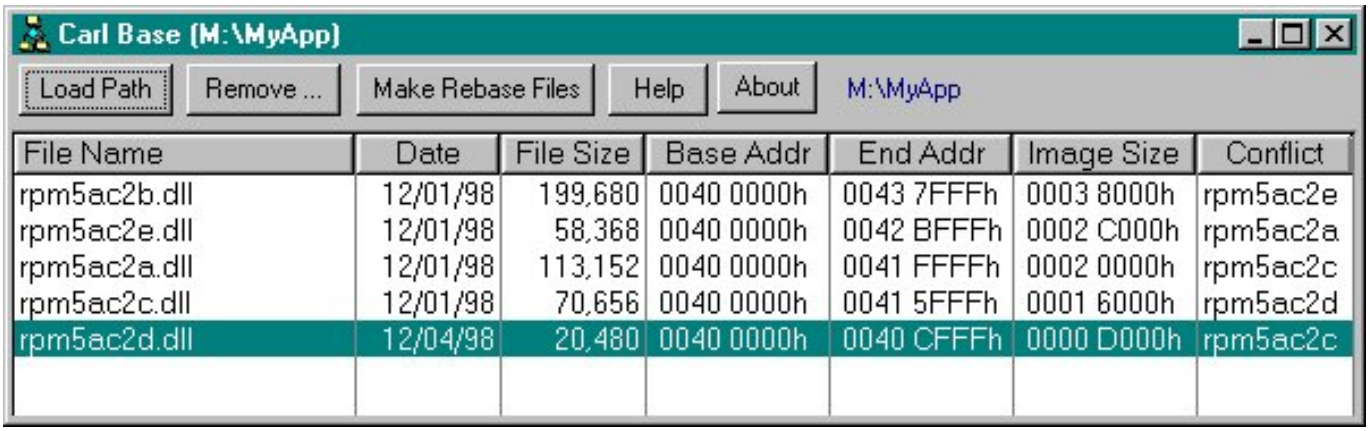
```
Echo Rebasing
Echo -b 07000000 is the starting base address
Echo Rebase automatically picks the next address based on DLL size
Echo -v Verbose
Echo
rebase -b 07000000 -v rpm5AC2a.dll rpm5AC2b.dll ↵
  rpm5AC2d.dll rpm5AC2e.dll rpm5AC2c.dll
```

One nice thing is if you get a GPF at an address in a rebased third party DLLs address space, you now know for certain which tool and which DLL had the problem. (Note: I've never had a GPF inside RPM.)

Using the CarlBase utility

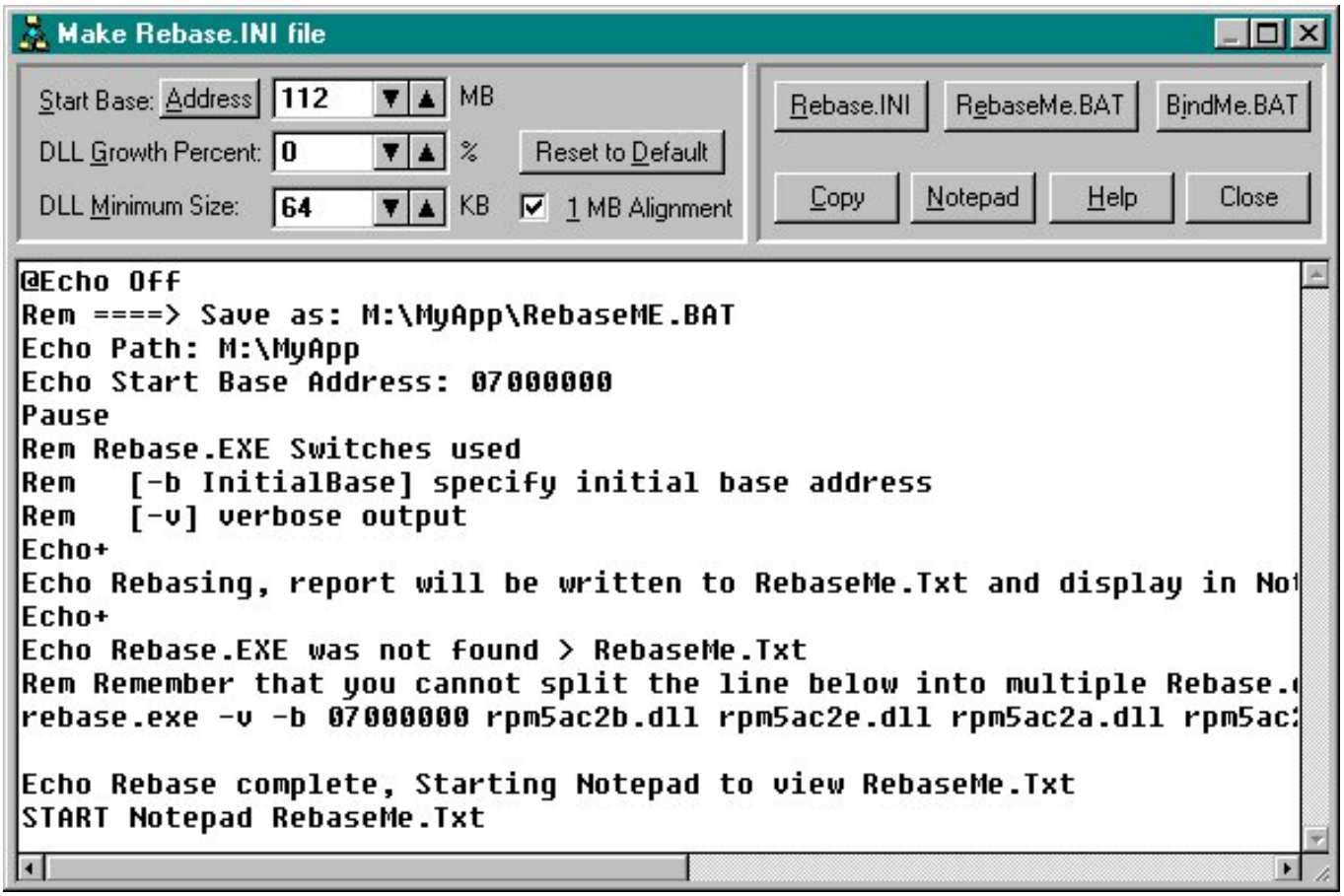
To make using Rebase.exe easier my CarlBase utility will write a batch file that calls Rebase.exe. You simply have to point CarlBase at the directory with the binaries and after a few keystrokes and mouse clicks you will have the batch file. The procedure is as follows:

1. Run the CarlBase utility
2. Press the Load button and select your App directory
3. Click on the DLLs that you *don't* want to rebase and press the delete key to remove them from the list
4. Press the "Make Rebase Files" button



File Name	Date	File Size	Base Addr	End Addr	Image Size	Conflict
rpm5ac2b.dll	12/01/98	199,680	0040 0000h	0043 7FFFh	0003 8000h	rpm5ac2e
rpm5ac2e.dll	12/01/98	58,368	0040 0000h	0042 BFFFh	0002 C000h	rpm5ac2a
rpm5ac2a.dll	12/01/98	113,152	0040 0000h	0041 FFFFh	0002 0000h	rpm5ac2c
rpm5ac2c.dll	12/01/98	70,656	0040 0000h	0041 5FFFh	0001 6000h	rpm5ac2d
rpm5ac2d.dll	12/04/98	20,480	0040 0000h	0040 CFFFh	0000 D000h	rpm5ac2c

5. Specify the Start Base Address (other parameters are not used)
6. Press the RebaseMe.Bat button
7. The RebaseMe.Bat file will appear in the text box
8. Copy the batch file and save it to disk in your App directory



Should you rebase with Rebase.exe?

Some people will ask "Should I use Rebase.exe and make binary changes to other peoples DLLs?" My answer would be an emphatic Yes! Windows DLLs are designed and built to be rebased. By rebasing a conflicting DLL you are just saving the Windows loader the trouble of doing it. If a DLL should not be rebased then it will be linked in a special way with the relocation table removed so that it cannot be

rebased.

I have been using the rebased RPM DLLs (that I rebased using the exact batch file above) and have not had a single problem. They have been running this way for over a year and on several thousand user PCs using every Microsoft 32-bit Windows. The primary purpose of the application is to produce reports, so I know that RPM is used extensively.

If a DLL is causing a conflict and it is based at one of the default addresses (`0040,0000h` for Clarion, `1000,0000h` for VC or `1100,0000h` for VB) then I would rebase it without any concerns. If a DLL has been assigned a reasonable address by the developer I will try to work around that address and not rebase the DLL. The primary benefit is that if a crash should occur inside the DLL the GPF dump will be more useful to the original developer since all the addresses will match his or her map file.

For example, the Clarion runtime C55nnnX.DLLs are all based in the `0080,0000h` to `0100,0000h` range. I never rebase these so a GPF address makes more sense to Alexey. There are several new DLLs that were built with Clarion and use the default `40,0000h` address: C55RTF, C55CR8, C55FINX, C55STAX. Since these DLLs get rebased by the Windows loader anyway I always rebase them to reduce the load time.

You should never rebase an operating system DLL. Microsoft very carefully assigns addresses and you should not mess with them or you could slow down Windows. You don't have to exercise any caution because Rebase.exe protects you from shooting yourself in the foot. The O/S DLLs are specially flagged and Rebase.exe will not modify them unless you specify the `-z` switch and bypass the protection.

A word of caution

Remember that Rebase.exe does make binary modifications to the DLL file that are not reversible, so you should keep a backup copy! The main time this becomes an issue is if your tool vendor issues you binary patch files instead of replacement files. You'll need to patch the originals, then save the patched versions for next time before rebasing.

A quick way to rebase your entire multi-DLL project

My previous article described how to modify your Clarion project so the linker would rebase your DLLs as it created them. This meant taking the time to modify every

App and performing a complete recompile. Instead you can use the Rebase.exe utility on your Clarion DLLs; this provides a simple and fast way to test your entire project rebased. Rebase.exe produces the identical results to doing it in the Linker and takes just minutes. However, I still think the Linker method is the best way so that your project is always rebased and your debug maps are correct.

Creating your own rebase utility

REBASE.EXE works by providing a simple shell around the API function `ReBaseImage()` which resides in the ImageHlp.dll. You can read about this function on MSDN; the prototype and call are very straight forward. You could write your own version of Rebase.exe by calling `ReBaseImage()`. Since Rebase.exe cannot be distributed you may wish to make your own. Eventually I'll probably add this feature to my CarlBase.EXE utility.

Success!

With REBASE.EXE you can eliminate all base address conflicts that cause an expensive Windows loader rebase. This will optimize DLL load times and DLL sharing, even when you're not the person who created the DLLs. In environments with a lot of DLLs and copies of DLLs this can make a big difference to performance.

Michael Brooks has applied the rebasing technique I've described in this series to his engineering applications. On a Dell PIII dual 733 Mhz 512 MB RAM server, Michael loaded 50 copies of one application. "Memory started at 423 Meg, and after 50 copies were running free memory was still 296 Meg as reported by the Win2K Task Manager. It seemed like the memory graph barely moved ! Needless to say I'm thrilled. Performance was identical to using [the application] with a local loopback on my NT4 development machine (but I use a cable modem and the servers are sitting right on "Telco Fiber"). This program basically has an APP frame, a browse, and a form window with 827 controls."

George Lehman, of Horizon Business Concepts, reports that this rebasing technique "chopped per-instance memory requirements nearly in half on a very large ClarioNET project ... which moved the whole project into the economically feasible arena. With literally only a couple of hours invested, including the time to read the articles and do before and after tests, I dropped per-instance RAM usage of my program from 34 to 18 megabytes. Load times for subsequent instances dropped approximately 30% as well. And all this without changing any actual program code AT ALL!"

I think that says it all. Next time I'll look at another super-easy-to-implement-with-no-downside optimization that pre-binds the imported functions lookup table to exported functions so that they no longer have to be looked up and bound at load time.

Resources

REBASE.EXE	For more information see the MSDN documentation
CarlBase	CarlBase is a utility I wrote to help you easily rebase your applications and explorer addresses in use. It will write you a batch file to run Rebase.exe. As discussed in my previous article it will also generate a correct Rebase.INI needed by my Rebase.TPL. You can download CarlBase from www.carlbarnes.com.
Process Explorer	Process Explorer is a free utility. It shows the base address at which each DLL was loaded and size of the DLL. (It defaults to showing handles, so to see DLLs you must select "View DLLs" from the View menu.) It does *not* show the original address specified by the linker. Thanks to Steven Gallafent for pointing out that this utility will highlight rebased DLLs if you check "Highlight Relocated DLLs" on the Options menu. This tool has many other features, is free and is only 77K.

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

Reader Comments

Add a comment

The CASE Statement Revisited

by Brian Staff

Published 2002-02-19

In the past, I have sometimes coded complex nested IF statements (which I find are hard to read) something like this:

```
IF condition1
   do whatever1
ELSE
   IF condition2
      do whatever2
   ELSE
      IF condition3
         do whatever3
      ELSE
....
END
```

I started to think that maybe I could use the CASE statement to code this type of logic. But, in all the Clarion code that I've seen, the CASE statements are usually used in this traditional way...

```
CASE condition
OF   value
OF   value
....
END
```

After a few experiments, I found that the CASE statement can also be used in this non-traditional way too.

```
CASE value
OF condition1
OF condition2
....
```

```
END
```

However, coding an "OF condition" is not always obvious. For instance, the following code will not compile

```
CASE true
OF   x=y
...
END
```

But the following will compile and does work:

```
CASE true
OF   CHOOSE(x=y,true,false)
...
END
```

So, I can now replace a hard-to-read IF structure like this:

```
IF x%2 = 1
   ?R2{PROP:Text} = 'x odd'
ELSE
   IF y%2 = 1
      ?R2{PROP:Text} = 'y odd'
   ELSE
      IF z%2 = 1
         ?R2{PROP:Text} = 'z odd'
      ELSE
         ?R2{PROP:Text} = 'all even'
      END
   END
END
```

with a CASE structure like this:

```
CASE true
OF   CHOOSE(x%2=1,true,false)
     ?R2{PROP:Text} = 'x odd'
OF   CHOOSE(y%2=1,true,false)
     ?R2{PROP:Text} = 'y odd'
OF   CHOOSE(z%2=1,true,false)
     ?R2{PROP:Text} = 'z odd'
ELSE
     ?R2{PROP:Text} = 'all even'
END
```

See the downloadable source for a complete working example.

Download the source

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

Brian, that certainly is an innovative use of CASE - I had...
FYI: choose(x%2=1,true,false) is equiv. to...
Further to Geoffs comment, the choose can be shortened...
Your approach to the CASE evokes the rich EVALUATE...
CHOOSE() is not the only option, any function that returns...
Carl said "CASE True is interesting, I doubt I'll use...

Please, User Dearest, Complete The Fields

by Steven Parker

Published 2002-02-20

Standard Clarion behavior for a required field left blank is, simply, to select it. No notification, no message, no nothing (a MSG in the status bar doesn't count). Some PCs beep; mine's not one of them.

Some people consider this less than optimally user friendly (haven't heard that buzz phrase in a while, have you?). Some of those people are users (remember them, the folks that indirectly cause us to be retained in employment?). The fact that some of them shouldn't be allowed within sight of a computer (or a telephone) is not germane.

That is, supposed you don't want to use the built in required field checking (and you uncheck the Required checkbox on the button's property sheet).

It is entirely possible to create a totally user friendly method of handling this. In the update form's Ok Button, `Accepted, Before Generated Code`, embed add the following (salt to taste):

```
If ~CUS:CustomerID
  MESSAGE('Dear, kindly, user, '&|
    '||Through no fault of your own, I''m sure, you '&|
    'have failed to enter an ID for this customer. '&|
    'This will make it most difficult for me '&|
    'to deal constructively with the information you have '&|
    'managed to enter.'&|
     '||Love,'&|
     '|Your dedicated software', |
     'Don''t Do That!', ICON:Exclamation)
  Select(?CUS:CustomerID)
  Cycle
End
```

Then, you get to do this for every required field on your form.

If you have two or three or four required fields, this is manageable, I guess. More than that … can you say "right!"?

I commend the `Incomplete()` function to your attention.

"The INCOMPLETE procedure returns the field number of the first control with the REQ attribute in the currently active window that has been left zero or blank" says the Language Reference Manual. Therefore the following code will handle *all* required fields on a form:

```
If Incomplete()
  MESSAGE('Dear, kindly user,'&|
     '|I''m sure it was entirely inadvertent on your part '&|
     'but you did manage to leave out a required piece '&|
     'of information.'&|
     '|I am going to show you which field it is and, '&|
     'then, I''m going looking for anything else you '&|
     'left out.'&|
     '|Regards,'&|
     '|Your efficient software developer', |
     'Fill It In, Fella', ICON:Hand)
  Select(Incomplete())
  Cycle
End
```

Well, that was easy.

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

Nice little tip. A Function that I had not...
wrong embed Steve.... ... "In the update form's Ok...

Clarion Magazine

Using Real Icons In The Listbox Header - Part 1

by Steffen Rasmussen

Published 2002-02-22

For quite some time now I've been working with the idea of having real icons in the header column of the list box. I've tried several different solutions about which I have written a couple of articles. But these solutions never really got it right. I still couldn't place real icons in the List box header, only substitutes in the form of ASCII characters.

If only I could give each header in the list box the same functionality as a button! So why not just use buttons to substitute each header column?

The problem with using buttons for each header column in a list box is, that in most cases the layout of the list box isn't static. You will have to take the following into consideration:

- Columns can change in width.
- The list box can change in width.
- The list box can contain more columns than are visible (horizontal scrolling)
- The first column from the right can have different widths depending on the amount of data in the list box. For example if there are more rows of data than visible, a vertical scrollbar will automatically appear on the right side of the list box.

Creating list box header buttons

Each column has to have a corresponding button to be used for the header. Just placing the buttons over each column header and resizing them is straightforward unless there are more columns than can be seen in the list box. I'll come to this later so for the time being I am going to stick to the basic list box, where all the columns are visible.

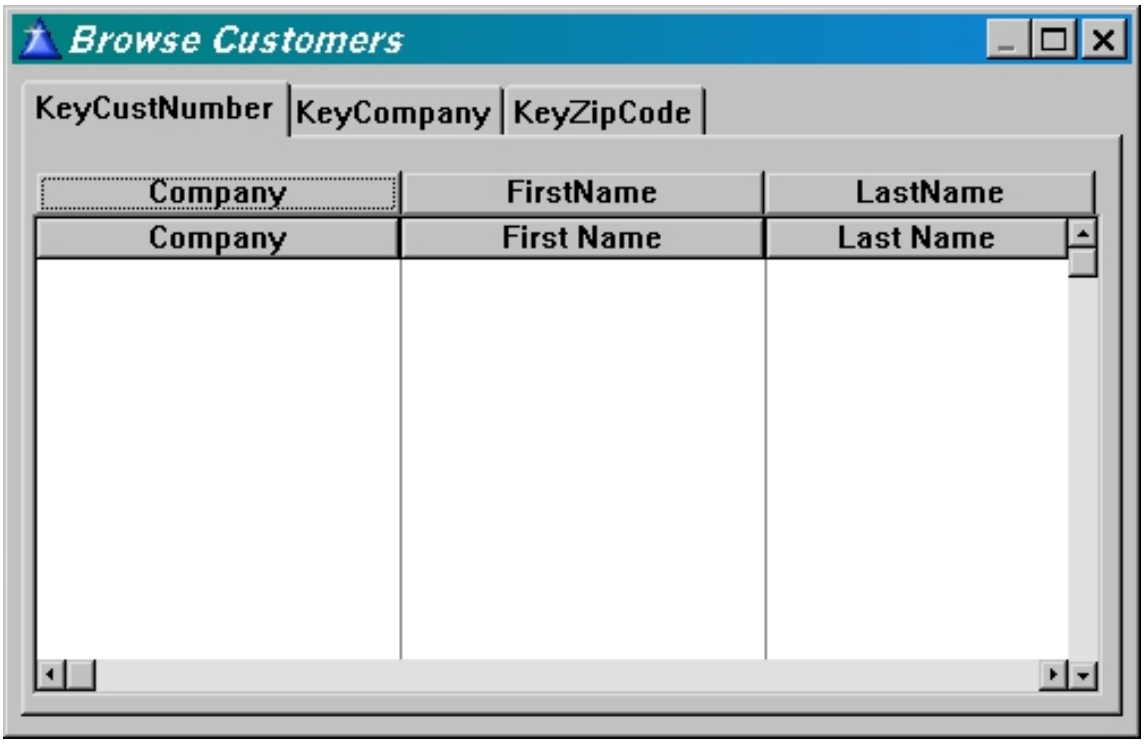


Figure 1. Placing the Buttons

As you can see in Figure 1 it looks as if the new buttons are just hanging there without any attachment to the actual list box. If you take a close look at the list box, you will notice that it looks as if the list box is placed within a panel control with an outer and inner bevel value of minus one. If you take a corresponding panel control and just place it around the buttons, it still looks unattached; on the other hand if it is resized around the whole list box including the buttons it will get a whole and consistent look.

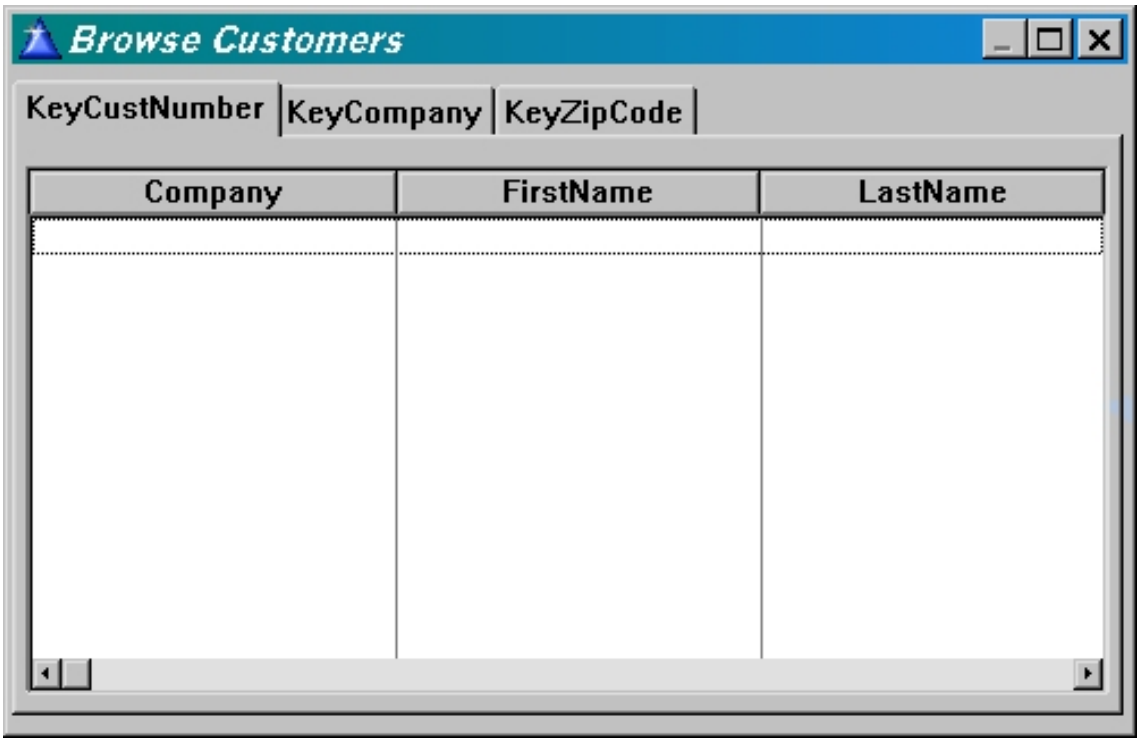


Figure 2. Integrating the buttons in the list box

> **NOTE**: In the above figure I have removed the original list box header just by deleting the header text of every column.

Although this approach does not look exactly as the original list box, it is pretty close. Now you can customize each column header in the list box in the same way as a button.

Next you will have to code the actual button integration into the list box, so the header always corresponds to the width and placement of the column underneath. At first this task seems straightforward, but there are a lot of coordinates which you have to track.

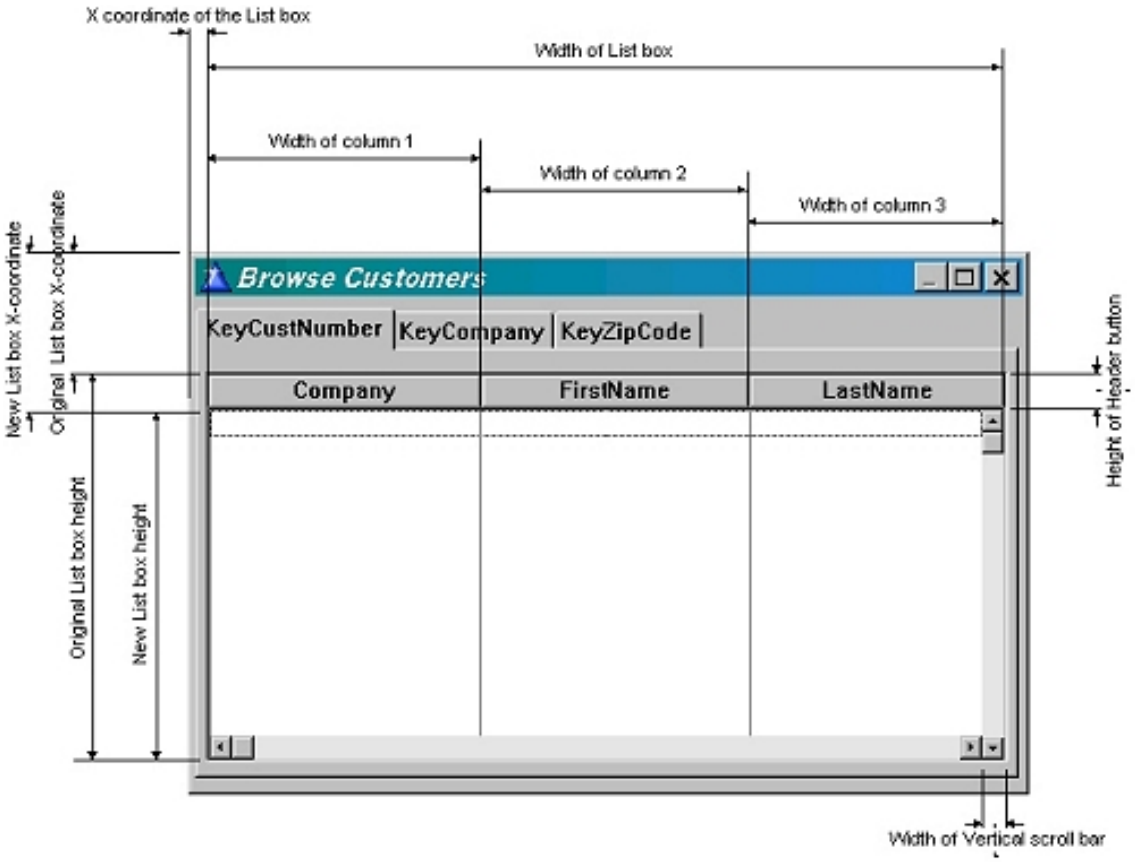


Figure 3. Different coordinates of the List box

Figure 3 should give you a pretty good idea of what task lies ahead. Lets start with the List box.

Sizing the list

The List box should be easy. It's all about keeping track of the x and y coordinates, and the width and the height. Let start off, with the panel that surrounds the listbox. The panel needs all of the x, y, width and height parameters of the list box in order to resize correctly when the window changes proportions. Now the panel and the list box are interchangeable. What I mean with this is that the original list box had a y coordinate and a height. These two parameters have to be changed in order to give room for the new header buttons, so the original list box doesn't change in height when the new header buttons are placed. It is a matter of keeping track of the original coordinates and the new coordinates in conjunction. I hope you are with me so far because I'm just getting started.

Now for the buttons, which are going to substitute for the header of the list box. For each button you will have to know all of the four parameters: x, y, width and height. In order to get these parameters, you will have to make a few calculations, which have to be saved and reused. To make this task a lot easier, create a dummy

button. This dummy button is going to be hidden at all times and won't interfere with the actual view of the list box. The dummy button contains the values of the x and width parameters of the column field which have focus. So how can a column field have focus? A column field has focus when the user resizes the column with. The y coordinate, on the other hand, is determined by the y coordinate of the panel plus 1 so that the button doesn't overlap the panel.

Now you just have to assign the parameters to the dummy button, with the PROP:Edit command:

```
?Browse:1{PROP:Edit,ColumnNumber}='The Dummy Button'
```

and then the GETPOSITION command is used to get the x coordinate and width:

```
GETPOSITION(DummyButton, x,, width)
```

You save the x and width position in a local variable, but the x coordinate is relative to the list box's x/y coordinates. What this means is that every object position within the list box is determined from the list box's x/y coordinate where x=0 and y=0. So if you want to get the button's x position, you need the x coordinate for the list box as well as the x coordinate for the column, which you add. Apart from this you also move the position one unit to the right so that the button isn't located right on top of the border of the previous placed button. Do this by adding 1 unit to the x coordinate as shown in the following line:

```
SETPOSITION(HeaderButton, 1+ListBoxX+ColumnX,ListBoxY-↵
   (PanelY-1),ColumnWidth)
```

Now you just have to loop through each column and assign each header button its new size and position, that is until the amount of data exceeds the maximum visible area in the list box. In this case you would most probably have a vertical scroll bar present. The vertical scroll bar changes the width of the last column in the list box. An obvious consequence of this is that the last header stops just where the vertical scroll bar starts:

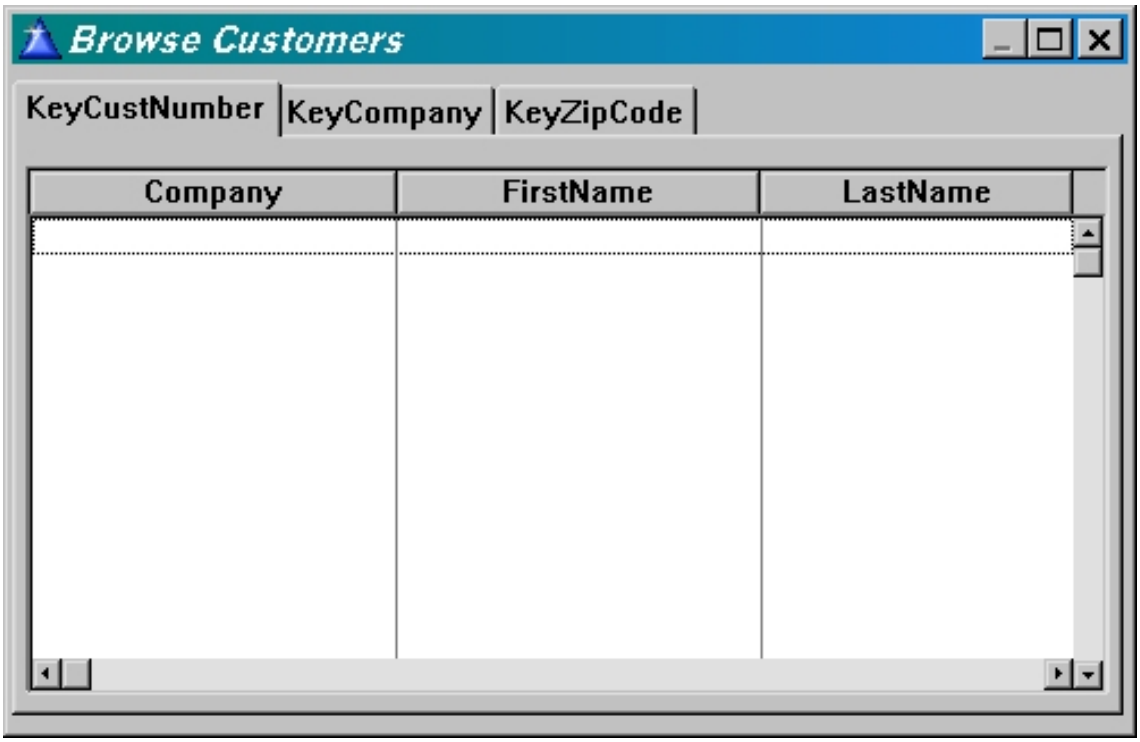


Figure 4. Including the scroll bar with the list box.

As you can see in the top right corner of Figure 4, this really breaks up the consistent look and feel of the list box, so you have to determine if the vertical scroll bar is present or not. For this use a local variable:

```
LOC:VScroll=?Browse:1{PROP:VScroll}
```

When the vertical scroll is present `PROP:Vscroll` have a value of one, otherwise it would be zero. Apart from this you also need the total width of the header buttons. When you loop through the code placing each header button, you add up the width of these header buttons and compare them to the total width of the list box, minus the width of the vertical scroll bar (if present). The width of the scroll bar is 6 units, but it isn't desirable to have a column which ends exactly where the scroll bar begins. Therefore I have chosen to give the width of the scroll bar a value of 12 units. Now it is just a matter of determining if the vertical scroll bar is present. If it is you just have to subtract the 12 units from the list box width, if not leave it as it is.

Determining whether you should subtract the 12 units or not is very easy in Clarion. All you have to do is to multiply the 12 units with the local variable `LOC:Vscroll`. This local variable contains 1 if the vertical bar is present and the resulting multiplication is 12. The other scenario is when the vertical scroll bar isn't present. In that case `LOC:Vscroll` is 0 and when you multiply this with the 12 units the result is still 0. Hopefully the following code example makes this clear:

```
IF ListboxMaxWidth < ?Browse{PROP:Width}-(12*LOC:VScroll)
  Button Width=Column Width
ELSE            !Last visible column in list box
  Button Width=ListboxWidth-ButtonX-2
END
```

The –2 takes into account that the button must not rest on either the button to the left, nor on the border of the panel to the right.

The last visible column in the list box isn't necessarily the last visible column. You could have an instance where there is a column right underneath the vertical scroll bar. If this happens you will end up width a tiny header button above the vertical scroll bar. To prevent this, all column header buttons, which are equal to or smaller then the vertical scroll bar, must be hidden:

```
IF (9*VScroll) =< (?Browse{PROP:Width}-ButtonX)-1
  HeaderButton I#{PROP:HIDE}=0   !I#=Column number
END
```

You are just about done with this small algorithm; there is just one last thing that has to be taken into account, and that is the possibility that the list box will have a horizontal scroll bar.

How does the horizontal scroll bar work? In a lot of instances it isn't possible to show all of the columns at the same time in the list box. Therefore some of them are "scrolled" out of view. What actually happens is that the columns which are not within the view are placed to the immediate left or right of the list box, depending on which side of the visible area they have been scrolled out of. When the columns are outside the visible list box area they are given a width of 1 unit and hidden. So when looping through this algorithm for the header buttons, which are outside of the list box, the code will give all a width of 1 unit and the specified height. To prevent this from happening you just have to hide all the header buttons that are outside the list box area.

Here is a smaller and less detailed version of the final algorithm:

```
GETPOSITION(?Browse,ListboxX,ListboxY,ListboxWidth,ListboxHeight)
SETPOSITION(?Panel,ListboxX-1,ListboxY-(PanelHeight+1),
| ListboxWidth+2,ListboxHeight+PanelHeight+2)
ListboxMaxWidth=0
LOC:VScroll=?Browse{PROP:VScroll}
LOOP I#=1 TO MaxNumber of Buttons   !I#=Column Number
  ?Browse{PROP:Edit, I#}=DummyButton
  GETPOSITION(DummyButton,ButtonX,,ButtonWidth)
```

```
    HeaderButton{PROP:HIDE}=1
    IF ButtonX<?Browse{PROP:Width}
      ListboxMaxWidth=ListboxMaxWidth+LOC:ButtonWidth+1
      IF ListboxMaxWidth<?Browse{PROP:Width}-(12*LOC:VScroll)
        SETPOSITION(HeaderButton I#,1+ButtonX+ListboxX,ListboxY%  |
        -(PanelHeight-1),ButtonWidth)
        HeaderButton I#{PROP:HIDE}=0
      ELSE
        SETPOSITION(HeaderButton I#,1+ButtonX+ListboxX,ListboxY- |
        (PanelHeight-1),(?Browse{PROP:Width}-ButtonX)-2)
        IF (9*VScroll)=<(?Browse{PROP:Width}-ButtonX)-1
          HeaderButton I#{PROP:HIDE}=0
        END
      END
    END
  ENDLOOP
```

You have to run this code on the initial start up of the procedure, and every time a change has been made to the list box. To do this it would probably be best to place the code in a routine from where it can be called when needed.

It is quite easy to make the initial call to `RoutineListboxHeaderResize` on procedure startup. But what about all the different kinds of list box changes? For this I have decided to use a timer to check the status of the list box every 20th of a second:

```
TARGET{PROP:Timer} = 5
```

Every time the timer is triggered the program looks for changes in the following parameters:

List box format	`PROP:Format`
Horizontal scroll position	`PROP:HScrollPos`
The x position of the listbox	`PROP:XPos`
The y position of the listbox	`PROP:YPos`
The width of the listbox	`PROP:Width`

The height of the listbox	PROP:Height

All these values are kept in local variables to which they are compared. For example PROP:Format contains the display format of the list box. Any changes to the width of any column in the list box changes the PROP:Format string. If this is the case, the following code assigns the new format to the variable and calls RoutineListboxHeaderResize:

```
IF LOC:ListboxFormat<>?Browse{PROP:Format}
   LOC:ListboxFormat=?Browse{PROP:Format}
   DO RoutineListboxHeaderResize
END
```

And so forth…

In this way the amount of processing is kept to a minimum because the actual work is only done if there is a change in any of these six parameters.

I could stop here, but really, what's the point of substituting the original list box headers with control buttons? First of all the idea was to be able to use real icons in the header but for this, real buttons aren't necessary. No, I also wanted to be able to initiate list box sorting when selecting a column header. And a template to do all the hard work would be nice. I'll demonstrate such a template next week.

Download the source

Steffen S. Rasmussen has graduated in Computer Science from Copenhagen Business College. Since then he has worked as a programmer, system technician and network administrator, and is currently IT manager. Clarion is a quite a new language to Steffen since his only been working with it since January 2000. But what better way to learn it than by trying to teach others! Steffen has also set up a web site to collect as many examples of different user interfaces as possible to inspire Clarion developers.

Reader Comments

Add a comment

Steve - been looking forward to an article like...
Thanks James - any comments,ideas or improvements are...

Clarion Magazine Begins Fourth Year Of Publication

by Dave Harms

Published 2002-02-22

Hard as it may be to believe, this February marks the beginning of Clarion Magazine's fourth year of publication. That's right – ClarionMag first hit the virtual newsstand in February of 1999. The first issue included articles on templates, OOP, Topspeed driver error codes, the ErrorClass, application debugging, the Clarion development environment, and more. It also featured a product review (Xplore templates) and the first part of an interview with Topspeed's then-president Roy Rafalco.

Since that first month Clarion Magazine has grown considerably. With three years worth of Clarion Magazine articles and two years worth of (freely available) Clarion Online articles, this site now contains over one million words of Clarion-related information. That's the equivalent of more than six 500-page computer books!

Clarion Magazine has grown a number of useful features over the past three years. You can now update your subscriber information online, perform complex searches on that million-word article base, add your own comments to articles, display articles in printer-friendly format, find articles using the topical index, and even automatically exchange future issues for back issues online (you'll be prompted whenever you attempt to read a back issue you haven't bought, and you have an unused issue, or issues in the case of double or monthly issues, left on your subscription). You can also buy all the back issues you're now missing - just go to the order page and choose the back issues only option.

Besides all of the functional changes, the magazine has also gone through a number of graphic redesigns. You can be sure that whether in appearance or features, ClarionMag will continue to evolve to better serve the Clarion development

community.

And don't forget the Clarion Magazine Sweeps! Now in its final seven days, this is your opportunity to win either an ETC-III conference registration, or a Compaq iPAQ (no purchase required). The Sweeps closes February 28th, 2002. You can enter by referring a friend, renewing your subscription, taking out a new subscription, or by sending in a handwritten postcard.

My thanks to the Clarion Magazine subscribers and authors, who have made the past three years of publication possible. I hope you look forward, as much as I do, to the many Clarion Magazine issues to come.

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

Way to go Dave! If you were to think back about why there...
Russ, Flattery will get you everywhere. Thanks for...

Using Real Icons In The Listbox Header - Part 2

by Steffen Rasmussen

Published 2002-02-26

In Part 1 of this series I began my quest for real icons in list box header columns with a lot of source code that showed how to achieve this using buttons in place of header columns. The code to do this can get a bit complex, and placing buttons for each column can be tedious and error-prone. To make life easier, I'll now show you how to create a couple of templates that do all the hard work.

My initial objective in creating this template was to have the same functionality in each column header as in a normal button. Therefore I have created a template where the prompts resembles the properties of a normal control button.

First of all you will have to make the template show all the columns of the list box. This is done by making sure that the template is associated with a list box, and you do that with the REQ(BrowseBox(ABC) attribute:

```
#EXTENSION(BrowseHeader,'Substitute headers with |
buttons'),WINDOW,PROCEDURE,REQ(BrowseBox(ABC))
```

In this way the template extension will be included within the folder for the particular list box in the Extension and Control Templates properties:

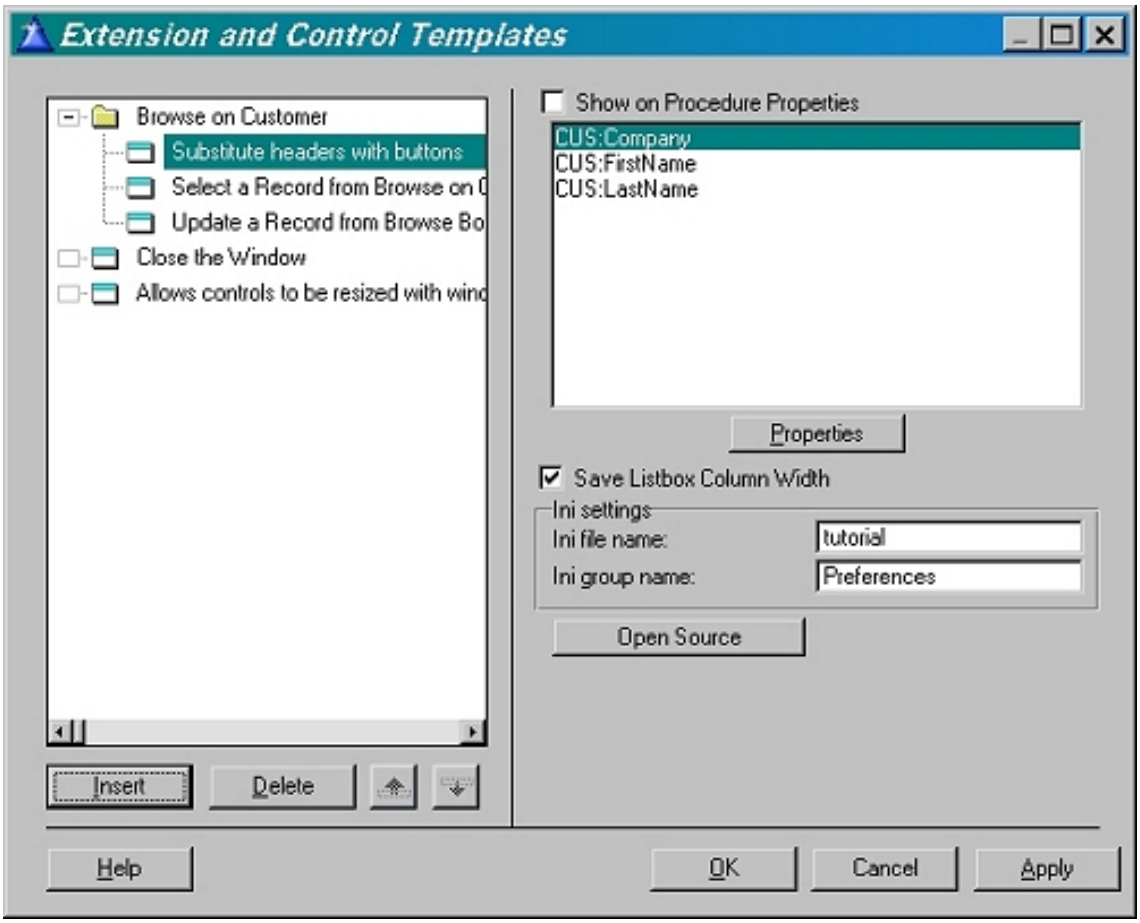


Figure 5: Extension and Control Templates properties

As you can see in Figure 5, the right hand template list box contains every column in the selected list box. You can also see the possibility to save the list box column in an INI file. I'll come back to this later.

In order to get the template to automatically show every column in the template list box, the template code has to prepare the list box and find every column:

```
#PREPARE
  #FIND(%ControlInstance,↵
     %ActiveTemplateParentInstance,%Control)
#ENDPREPARE
```

Every statement within the `#PREPARE` and `#ENDPREPARE` is a command to fix multi-valued parameters to the `#PROMPT` or `#BUTTON` statements. In this case it is the `#BUTTON` statement you want to fix. The `#FIND` statement fixes all multi-valued parent symbols to values that point to a single child instance. Here the `%ControlInstance` is the list box number for which the template is associated; the `%ActiveTemplateParentInstance` is the instance numbers of all control templates used in the procedure; and the `%Control` is the equate label of all controls in the window.

Now all the instance numbers (%ActiveTemplateParentInsatnce) limited by the %Control(?Browse:1) have been associated with the %ControlInstance, and in this way the %ControlField of the list box is automatically populated as you can see in figure 5. You use all these template commands to create an auto populated list box, by means of %ControlField:

```
#BUTTON('Header Button Properties'),↵
   FROM(%ControlField,%ControlField),INLINE
```

Next you have to create the template menus for assigning the button properties for each button header column. Since these menus resemble the template menu for the button I'll leave it up to you to study the template to see its construction. Although every template property from the button is included in the menu, they are not all used or tested with this template. This is also left to you.

Initially you will have to get all the data from the list box when the procedure is executed for the first time in the program session. This includes the list box's x, y, width and height as well as the properties for each column header, especially the header name. The header name is very important because each new button you create will contain the name of the column. The original name is going to be deleted in order to remove the original header from the list box. Apart from removing the header the list box also needs to be resized so the actual height of the list box does not change when the code creates and places the new header buttons.

I have made it possible for you to define the height of each header button. In most cases the height is the same, but in order to support this feature the code must determine the height of the highest button, so the list box can be resized correctly. The template does this by looping through the height parameter for each button in the template code. When the maximum height is determined it is subtracted from the original list box height to make room for the buttons. Also the height is added to the y coordinate of the list box so you reposition the list box right underneath the header buttons. When you have set and saved the initial parameters the RoutineListboxHeaderResize procedure is called to update all the new settings.

Next you have to dynamically create the controls needed. These include the panel, the header buttons and the "dummy button" called LOC:HeaderProperty. By creating these three controls dynamically you can concentrate on the size and position of the actual list box, just as you usually do when you create it in the Clarion window editor.

The template code only needs to create the panel and the dummy button once. On the other hand it will have to create a header button for each column in the list box. For this you use the #FOR command which loops through every template list box instance that contains a column. The #FOR command loops through the list box in sequence so the first column is selected first, the second, second, the third… and so forth. Although #FOR loops through in sequence there is no way you can tell which column number is used unless an extra template variable is introduced. Lets call it %ColumnNumber:

```
#SET(%ColumnNumber, 0)
#FOR(%ControlField)
  #SET(%ColumnNumber, %ColumnNumber+1)
```

As you can see %ColumnNumber is reset to zero before the loop is initiated. For each iteration you determine the column number, by adding 1 to the %ColumnNumber variable. The %ColumnNumber is not defined by any template prompts therefore you will have to initialize it in the template #ATSTART section:

```
#ATSTART
    #DECLARE(%ColumnNumber, LONG)
#ENDATSTART
```

In order to dynamically create controls in Clarion, you use the CREATE command:

```
CREATE (control, type [, parent] [, position])
```

The control is, in this case, a local variable called LOC:HeaderButton%ColumnNumber. The type is the type of the control, and for this particularly task you will want it to be a button (CREATE:Button). The parent is used when the control is within another control, for example a tab within a sheet. In this case, you won't use it. Neither will you specify the position of the newly created controls because these are taken care of by the RoutineListboxHeaderResize routine. Since a variable is needed for every control, you will need to use the #FOR command once again in the templates #AT(%DataSection), which is where you place all the variables needed for this particularly template:

```
#AT(%DataSection),PRIORITY(3000)
LOC:HeaderProperty   STRING(20)
LOC:HeaderPanel     STRING(20)
#SET(%ColumnNumber, 0)
#FOR(%ControlField)
    #SET(%ColumnNumber, %ColumnNumber+1)
LOC:HeaderButton%ColumnNumber STRING(30)
LOC:HeaderName%ColumnNumber STRING(30)
#ENDFOR
```

```
...
#ENDAT
```

After you have created the variables for the buttons it's time to dynamically create the actual header button:

```
LOC:HeaderButton%ColumnNumber |
  =CREATE(LOC:HeaderButton%ColumnNumber
,CREATE:Button)
```

As you can see the button is created and assigned to the local variable, `LOC:HeaderButton%ColumnNumber`.

This variable is a field equate that references the actual button and all of its properties. Most of the properties in this template are assigned by you in the template menu. For example if you check the hide control, the property of this control is saved in the template variable `%HeaderHide`. Assign this property to the created button this way:

```
LOC:HeaderButton%ColumnNumber{PROP:Hide}=%HeaderHide
```

You use the same approach for all of the variables that has to be set for the particular button.

When a control is dynamically created it is initially hidden, so in order to make it visible for the user the control has to be unhidden. In this template the hiding and unhiding of the controls are done in the routine `RoutineListboxHeaderResize` created earlier.

Now since you have reached the point where all the controls have been created and assigned their respective properties, it is time to delete the original list box header. Almost. Before this is done it is a good idea to save the name of each column header in yet another local variable, `LOC:HeaderName%ColumnNumber`.

```
LOC:HeaderName%ColumnNumber |
  =%Control{PROPLIST:Header,%ColumnNumber}
```

The reason for this is that in order to save any changes the user might have made to the column widths in an INI file, you also have to save the header name before it is deleted from the original list box. If you just deleted the text from the header, the next time the user opens the procedure the buttons wouldn't contain any text. Instead, when the user closes the procedure, you save the user's list box preferences by recreating the original header in the list box from the previous saved

variable, and then saving the new properties to the specified INI file:

```
#AT( %WindowManagerMethodCodeSection, ↵
   'Ask', '()'),PRIORITY(5001)
#SET(%ColumnNumber, 0)
#FOR(%ControlField)
    #SET(%ColumnNumber, %ColumnNumber+1)
%Control{PROPLIST:Header,%ColumnNumber}|
   =LOC:HeaderName%ColumnNumber
#ENDFOR
    PUTINI('%IniGroupName','%Procedure %Control Format'|
      ,%Control{PROP:FORMAT},'%IniFileName.INI')
#ENDAT
```

Now that the name of each column header has been saved it really is time to delete the original list box header:

```
!Delete header text
%Control{PROPLIST:Header,%ColumnNumber}=''
```

The last step is to call the `RoutineListboxHeaderResize` so all the controls just created can be positioned correctly.

The remaining piece of the puzzle is triggering a call to source code when the user clicks on a created button control. I'll explain how to do that next week.

Download the source

Steffen S. Rasmussen has graduated in Computer Science from Copenhagen Business College. Since then he has worked as a programmer, system technician and network administrator, and is currently IT manager. Clarion is a quite a new language to Steffen since his only been working with it since January 2000. But what better way to learn it than by trying to teach others! Steffen has also set up a web site to collect as many examples of different user interfaces as possible to inspire Clarion developers.

Reader Comments

Add a comment

This is way cool! Partly back online, Tom

A Closer Look At Required Fields

by Steven Parker

Published 2002-02-28

Carl Barnes quite rightly took me to task for my references to the Ok button in my article First Field, Required Field. He points out that "The Ok button you refer to at times probably should be called the 'Save Button'."

I had, in fact, assumed that the procedure I was discussing had been created from the Form template and not the Window template. Indeed, as I look back, I have *always* made this assumption when referring to the Ok button.

But Carl is right: there is a significant difference between the Ok button provided by the Form template and that provided by a Window template procedure when you select Window with OK and Cancel (as in Figure 1).

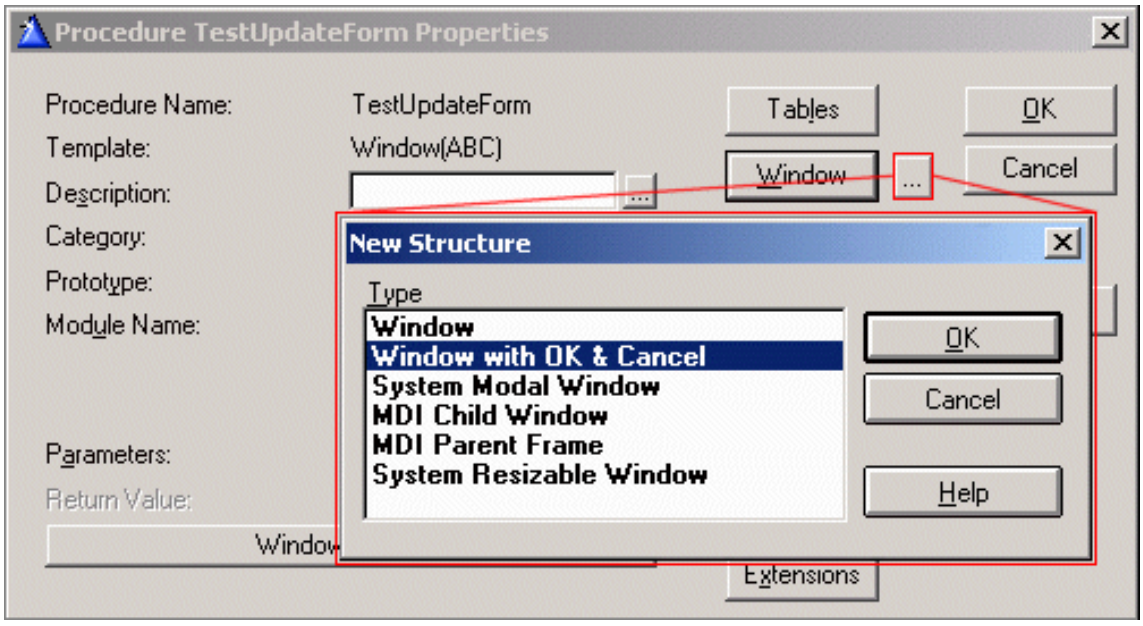


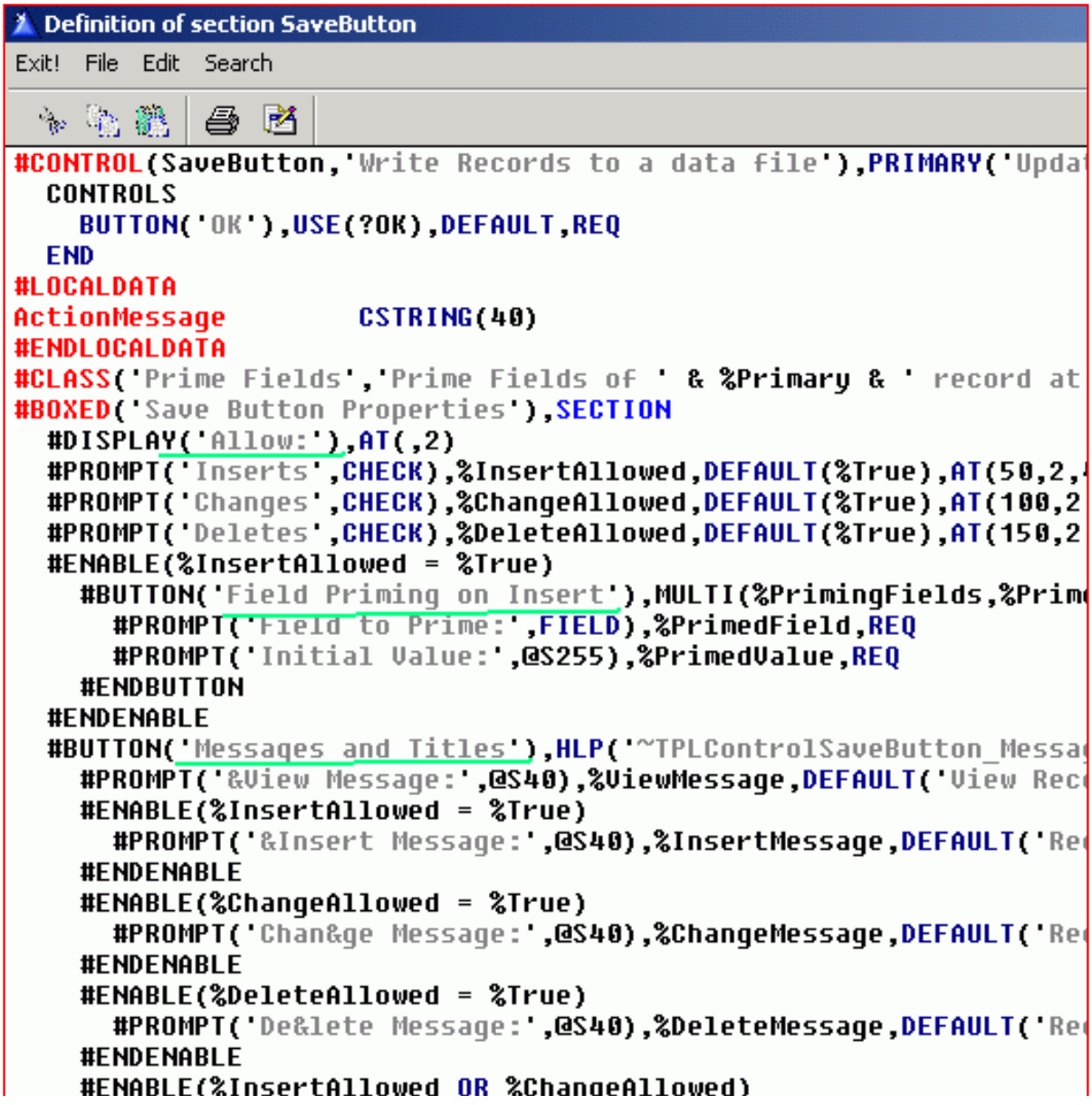
Figure 1. Window Template: select type of window

The Ok button on a Form is actually a control template, the Save Button template. You don't "see" this because the Form template pre-populates it. The Ok button on a Window is just a button. It doesn't do anything. Neither, for that matter, does the Cancel button on such a Window. It is the Save Button template that changes a (mere) window into a form.

One quick look at the template definition provides convincing evidence. Choose Setup|Template Registry from the Clarion IDE and search for Save, and you will immediately notice that the Save Button template is described as "Save Button – Write records to a data file."

Interesting. But there's more.

If you look at the template code (part of which is shown in Figure 2), you will notice all those things you tend to associate with forms.



```
    #ENDENABLE
    #ENABLE(%InsertAllowed OR %ChangeAllowed)
      #PROMPT('On Aborted Add/Change:',DROP('Offer to save change
    #ENDENABLE
    #PROMPT('Field History Key',KEYCODE),%HistoryKey
    #ENABLE(%DeleteAllowed = %True)
      #PROMPT('When called for delete:',DROP('Standard Warning|Di
    #ENDENABLE
    #ENABLE(%InsertAllowed = %True)
      #PROMPT('After successful insert:',DROP('Return to caller|I
      #ENABLE(%AfterGoodInsert = 'Insert another record')
```

Figure 2. Save Button: Part of the template

You will see the check boxes for allowing inserts, changes and deletes. You will see the Field Priming button and the Messages and Titles button. The option for recursive adds is there and so are the options for what to do when a user cancels. Yes, virtually everything associated with a data entry form.

To really satisfy yourself, create a procedure with the Window template and select a plain window. Examine all the properties you can from the Procedure Properties window. This won't take long, there aren't many; in fact, except for a single datum, there aren't any. Then populate a Save Button from the window formatter's Populate|Control Template list. Examine the properties again.

Suddenly a primary file declaration is required and all of the extensions and prompts you expect on a Form are present (except one, Cancel the Current Operation, which is populated by the Cancel Button control template).

Checking Required Fields

Functionally, the most important thing the Save Button template does is insert this code in the `INIT` method (priority 7500):

```
SELF.OkControl = %SaveControl
```

Regardless of the `Field Equate Label` you assign to the control on the window (by default, it is `?OK`, pretty much as you would expect but you can change it to anything you want, say `?Milton`), the procedure knows it has an Ok-type button:

```
SELF.OkControl = ?Milton
```

Now, when this button is pressed, an `Event:Accepted` is generated and handled in the `TakeAccepted` method (this is the code from `PARENT.TakeAccepted()` with some commentary added):

```
WindowManager.TakeAccepted PROCEDURE
I LONG,AUTO
A SIGNED,AUTO

  CODE
  !assign Field Equate Label (FEQ) to local variable, A
  A = ACCEPTED()
  !if there is a toolbar
  IF ~SELF.Toolbar &= NULL
    SELF.Toolbar.TakeEvent(SELF.VCRRequest,SELF)
    IF A = Toolbar:History
      SELF.RestoreField(FOCUS())
    END
  END
  LOOP I = 1 TO RECORDS(SELF.Buttons)
  !if it's a button, set Response
    GET(SELF.Buttons,I)
    IF SELF.Buttons.Control = A
      SELF.SetResponse(SELF.Buttons.Action)
      RETURN Level:Notify
    END
  END
  !if this is the Ok button
  IF SELF.OkControl AND SELF.OkControl = A
    SELF.PostCompleted
  END
  RETURN Level:Benign
```

And the `PostCompleted` method is called. `PostCompleted` is where AcceptAll (non-stop select or non-stop mode) is set:

```
WindowManager.PostCompleted PROCEDURE
  CODE
    IF SELF.OriginalRequest = ChangeRecord OR |
       SELF.OriginalRequest = InsertRecord
      SELECT()
    ELSE
      POST(EVENT:Completed)
    END
```

As I discussed in First Field, Required Field, once `Select()` is issued, the Clarion runtime library (RTL) processes each control on the window by generating an `EVENT:Accepted` for it (this is what the LRM says; it may be somewhat simplifed - if so, that is an unnecessary complication for present purposes). If a field has invalid data (e.g., Must be in file or Must be in range, etc., and isn't), non-stop mode is interrupted. If `Window{Prop:AcceptAll}` is explicitly set to 0 (zero) or a blank required field is found, non-stop mode is interrupted. You can also interrupt non-stop mode by issuing a `SELECT(?SomeField)` where `?SomeField` is the field equate of the field in question.

Otherwise, all `Accepted, Before Generated Code` and `After Generated Code` embeds are executed. (N.B.: because `TakeAccepted` is a derived method, Before and After simply mean before/after the `PARENT.TakeAccepted` code, shown above for the typical, i.e. non-control template, controls on a window.)

Finally, the Language Reference Manual states, "When all controls have been processed, `EVENT:Completed` to the window."

Thus, every embed executes at least once and some execute more than once.

Having every embed execute at least once ensures that validations, required entries and any calculations or other checks you need to perform *will* get done, regardless of how many fields the user skips. Very convenient, this. And, if there is code that you *don't* want executing except when a field has been explicitly changed, you can bypass it by testing for AcceptAll mode:

```
If ~0{Prop:AcceptAll}
  !your code
End
```

The actual sequence of events in `TakeAccepted`, when pressing the Ok button (ok, the Save Button Ok button) is:

```
TakeAccepted, Before CASE ACCEPTED()
Ok Button, Before Generated Code
TakeAccepted, Before Parent Call
    (TakeAccepted, Parent Call here)
TakeAccepted, After Parent Call
Ok Button, After Generated Code
```

But, what is *really* important is that this sequence is repeated for every control that has code in its `ACCEPTED` embed (substituting the control's `USE` variable for Ok button).

In the sample app (made with Clarion 5), select the Form type procedure menu. The update form has various stopping points to show you all of this in action.

Where the trouble starts

When a Clarion developer decides to use the Window template and uses the with OK and Cancel as his or her default, the trouble begins.

The trouble is that none of the really important things happen as one would expect when using the Window with OK and Cancel. You will not know this until after

you've compiled and started testing your app.

In the sample app, I have an update form created from the Window template (using Window with OK and Cancel). From the Browse menu, select Window with Ok & Cancel. You won't like it.

At first, things appear to be ok. If you press Insert, the "form" comes up empty, as expected. If you press Change (assuming already you have some records in the file), the form comes up with the data for that record. Even `GlobalRequest` is set correctly.

So far, things appear normal. This is because the browse has set `GlobalRequest` and has prepared the record buffer (cleared and primed for an add; retrieving the underlying record for a change or delete). For a more detail discussion of the ways forms are called, see Calling Form Procedures (this article is a bit dated on the autoincrementing information, but otherwise current).

One minor difference occurs if you press Delete: the form appears, DOS style, instead of the default Are-you-sure type message.

But the situation deteriorates rapidly when changing or inserting a record and pressing Ok. Required fields, if blank, are not selected. You certainly would expect them to be.

Lookups, if required, are not called. And, worse, not only doesn't the record save, the window won't close. It won't close if you press Cancel either. Only Esc will close the window and, of course, that is self-defeating in a data entry form.

The buttons supplied by the template just don't do anything, nothing at all. There is no file I/O, no field checking, no `STD:Close`, nothing.

Checking the Required checkbox on the Ok button's property sheet will enforce required field checking. And, it is possible to embed file I/O (in the sample app, check out the Modified version of the Window procedure and the notes in the Ok button's `Accepted` embed).

But, getting a window to work as a form in this way just isn't worth the effort.

Am I advocating never using the Window template for an update form? No. I *am* saying that if you do use the Window template for an update procedure, make sure you use a plain window and the Save Button and Cancel Button templates. But, of

course, Window + Save Button + Cancel Button = Form Template.

So, you might just as well start with the Form template. The mystery, really, is what would be a legitimate use for the Window template as a data entry tool?

Secondary forms

Sometimes there are too many fields to place on a single window. Sometimes fields are grouped according to a logical division of data types. Spreading fields so that not all are displayed simultaneously makes the data easier to digest. Sometimes the user wants multiple screens and sometimes it just looks better.

A very common requirement is for a single window to show the majority of the data and less frequently viewed data to be on another page. The Clarion templates (ABC and Legacy) provide two distinct ways to spread entry fields across multiple (virtual) windows: tabs and wizards. But, many Clarion developers do not like these alternatives for design and aesthetic reasons. So they use windows for these supplementary data displays.

There are a number of issues in using secondary windows. I will cover those in a separate article; they *are* significant. For the moment, let me confine myself to handling required fields on these windows.

If you are going to use a secondary window, you must use the Window template and you cannot use the Save Button template. If you use a Form or a Save Button, the record is saved when you press OK. This also means that the Cancel button on the primary form is meaningless. I don't think I need to argue this.

It would also cause a changed by another station error if the user presses OK on the secondary window and then OK on the main data entry form. Not good. So, a Window with OK and Cancel is the template of choice for a separate, secondary window.

Other considerations aside, you can use `Incomplete()` or check the `REQ` checkbox on the Ok button's properties worksheet. And, because no required field checking is provided by default, you must do one or the other.

Summary

Understanding how required fields are implemented and how the ABC template enforce them isn't very sexy. But failing to understand how this happens can burn

you badly. Understanding can make your life much easier.

Whether you press the Ok button on a Form or use the Save Button template on a Window, remember that `Prop:AcceptAll` is true and your procedure goes into a loop, generating `Event:Accepted` for each control on the window. Knowing this makes invoking or by-passing the standard handling a piece of cake.

You know, this loop thing is really getting out of hand.

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