THE CODEBOOK NEWSLETTER

September-October 1997

Volume 1, Issue 3

TABLE OF CONTENTS

Editor's Comments

Charles T. Blankenship

Well, another DevCon came and went. I hope everyone had a great time learning about this wonderful language in which we are fortunate enough to develop. Maybe next year I'll be able to go and meet everyone ... I desperately want to put faces to all of your names.

The first article in this issue concentrates on a utility that I use daily in my Codebook development efforts and eradicates a problem that has plagued me since I learned to spell Codebook. Almost without fail, I forget to define the proper include file for the new, application level classes I create. Since I never forget to use at least one Codebook defined constant in each one of these classes, I always get errors during the first run of the new software. This sends me scurrying to define a couple of kabillion include files for my new classes. After I defined about 20 of these little skudders and rubbed the kinks out of my mousing finger I realized that no matter when I define them this activity takes the same amount of time.   I only recognized the magnitude of that time when I had to define them one after the other. I therefore created the utility presented in this issue that can define all kabillion in seconds, either interactively and safely or all at once without interruption. I hope it saves you as much time as it does me.

After writing both the utility and the article, I shared it with the associate editors, one of which is Ed Leafe. Lo and behold he had written a utility that did the same thing, named it the same but approached things a little differently. The critical note here is that our collaboration resulted in a utility that is better than either of the two originals. This is the exact synergistic relationship that I was hoping to achieve with The Codebook News.

Now that we have both The Codebook News AND Ed's new Codebook forum I have high anticipation that more developers will bump heads, exchange ideas and make the Codebook experience better than we have been experiencing in a singular, sheltered development environment.

The second article is from Jose Constant and he describes how to make the cBizObjMaint form, with its view parameters on page one and the grid on page three, work with eBizObj. Although there are many developers out there that view this form as an example of how NOT to design a user interface, I (as well as Jose obviously) still find a use for this venerable old workhorse. There are a few considerations for this approach however. Ed himself identified them quite nicely ... so, I'll let him tell you, from Ed to you with love <s> ...

> "Well, I know you all know how I feel about the whole MaintForm issue <g>, so I won't go into that. My only concern with José's article is the practice of calling eBizObj 'cBizObj', and adding a refBizObj to non e-classes. While it may work sometimes, it makes it less likely that changes to future versions will be as smooth as they should be. The recent change I made shifting the oSessionEnvironment object from the bizobj to the form, as well as the new registration methods in the eBizObjForm mean that you will be stuck with version 1 of eBizObj. The form and bizobj classes are tightly intertwined, and messing with that is bound to cause problems."

> "Perhaps adding a few excerpts from this letter to the article might be a good idea so that users will be aware of the potential for problems. Personally, I would think that making a copy of the cBizObjMaintForm class and redefining it to inherit from eBizObjForm would be a better way to go."

I there ... I'm back now ... I must apologize to everyone for my apparent absence from the Codebook community for the past couple of weeks. This newsletter as well as all of my other commitments and responsibilities have fallen victim to servicing clients. The good news resulting from all of this work is that I now have a plethora of new classes that I can dump on you. Some of the upcoming classes are

- Contained Controls and their builders (October, 1997)
- The DE Foreign Key Combo Box (data environmentally aware foreign key combo box, October 1997)
- Encrypted Contained Controls (the data source stays encrypted and is decrypted for viewing and data entry, November 1997).
- The Messaging class library (November 1997)
- The Trapped Error Message class library (November 1997)
- And, last but certainly not least, a full application security system that offers interface object level security, process security, full user security, group security, security specific to a particular user, and many many more features. This security system allows an administrator to control each and every aspect of the application. The information to support this is stored in a separate directory in a separate database container and is 100% encrypted on disk. It is only decrypted in memory when it needs to be used. (December 1997, January and February, 1998)

Thanks again to the associate editors, Ed Leafe, Jose Constant, Frank Cazabon and Kevin Emmrich. The participation of these gentlemen in this process continuously proves the power of synergy. Thanks guys.

Charles T. Blankenship is president of Software Assets of Virginia, Inc (SAVI), a computer consulting firm that specializes in developing mission critical Visual FoxPro based applications using Codebook technology.. Phone: (757) 853-4465, Internet: ctb@savvysolutions.com, Web Site: ***www.savvysolutions.com****, CompuServe 76132,2575*

Class Library Include File Definition Utility

Include files in the Codebook framework, while necessary as well as beneficial, can be a real pain. This utility automates the process of defining the include file for classes in your class libraries.

Charles T. Blankenship

Introduction

Recently I was defining a bunch of application level classes at a client site. After completing this task, I ran the application and realized that, once again, I had forgotten to include the APPINCL.H file into the class definition ... of *every* class definition. Well, about 200 mouse clicks later I was completed ... and depleted from the tedium. It was then, while rubbing my sore mousing finger, that I decided to write the utility that would prevent me from ever having to do that again.

Design Requirements

This utility had to allow me to populate classes with include files quickly while offering the following capabilities ...

1. Identify the project and have all class libraries associated with the project included in the processing ... this means both application and framework level classes
2. Process only those classes that do not, as of yet, have an include file defined ... i.e., add the include file to only the ones I forgot during the day's processing.
3. Provide the option to use the utility cautiously or boldly. Cautiously requires a developer grant authorization before every replace. Bold users may elect to let the processing run without intervention.

The Preliminaries

Structure of the VCX files (class libraries)

I had several problems to deal with writing this utility. This first was determining where to find all of the class libraries (.VCX files) in my project. The second involved determining how to identify the specific classes inside the class library itself.. Finally, once the class definition was found where was the include file information stored?

Finding the physical .VCX files that are a part of your application is easy and can be accomplished using one SQL-SELECT statement executing against the project file itself. One kinda like this ...

```
SELECT name FROM ( tcProjectAlias ) WHERE Type = "V" INTO CURSOR crsVCX
```

By extracting the **`name`** field for each record in the project file that had a type of "V" we could easily identify each and every class library associated with the application. Thanks Ed.

Finding the class definitions inside the class library file was the next hurdle. *Something* must uniquely identify these class definitions. The reason I say this is because when you click the "+" sign next to the class library, you see a list of classes contained in that class library. It only takes one peak into a .VCX to realize that there is not a one to one correlation between records in a class library table and the number of class definitions contained therein. In one class library I have there are 63 records in the .VCX but only two class definitions visually presenting themselves in the project manager. Finding how these class definitions are defined required only a small amount of hacking.

Just as an exercise, open one of the class libraries provided with this article by issuing the following commands **`USE AFORMS.VCX`** ... ensure you include the .VCX extension or you will get an error message. This opens the class library just as if it was a regular .DBF. Next issue the **`BROWSE`** command. Position the browse window so the **Reserved1** field is visible and scan the class library file for an indication that something is contained in this field ... look for **Memo** vice **memo**. When you locate a record where the **Reserved1** field contains information, double click on that field. It should contain the text string **Class**. Congratulations, you just found the header record for a class definition that is contained in the class library.

Now that we know how to identify the header records for class definitions, finding the column that contains the include file is only several clicks of the mouse away. Double click on the **Reserved8** field and revel in your accomplishment!

The final piece of information to find is the actual name of the class definition. This information is contained in the **Objname** column. This will come in handy when informing the developer which class definition is about to have an include file defined.

Include File behavior

The final hindrance involved determining what should be placed in the **Reserved8** field. Notice that when you examine the include file through the project manager, you get a fully qualified path to your include file. As an example, the include file for my **maintoolbar** class definition is ...

```
"e:\codebooknewsletter\volume1issue3\include\appincl.h"
```

However, when I examine the class library file itself (ATOOLBAR.VCX) and find the record that stores the **maintoolbar** class definition header information I find in the **Reserved8** field is

```
..\include\appincl.h
```

It is very obvious that the actual include file path stored in the class library is being augmented by some additional information. This additional information is provided by the project itself. There is a very good reason for doing this and that reason is to assist in keeping a project portable over drives and locations. FoxPro stores the location of the include files relative to the location of the classes in which they are used. As an example, my project's directory structure looks like this (partial listing)

```
MRS_APPS
        LIBS
        INCLUDE
```

Where MRS_APPS is the directory in which the project is stored, LIBS is where the class libraries are stored and INCLUDE is where all of the application's include files are located. It is now easier to see how to determine what to put in the **Reserved8** field for a class definition ... the *relative* path to the location of its include file NOT the full path definition with Drive and Path information hard coded. The include file is back one directory (**..**), in the **INCLUDE** directory, and its name is **APPINCL.H**. This also implies that you should keep your include files in their standard locations ... it just makes things easier.

Using SETINCL.PRG

This program is designed to operate from the CDBK30\COMMON30\UTILS subdirectory. So, when you extract this program from the ZIP file, copy it into that directory. When you want to use it simply execute the command DO STARTCB (so the FoxPro path points to the UTILS subdirectory) and then execute the command DO SETINCL. The only thing you must provide to this program is the identify of the project file you want it to execute against. Do this in the GETFILE() dialog that appears, press OK, stand back and don't get any on ya'.

The signature of this program is designed to make it process very quickly or interactively (slower, with more caution). Its structure is as follows:

```
=SETINCL( [<cReplaceAllIncludeFiles>] , ;
          [<cCautious>]                 )
```

The < cReplaceAllIncludeFiles > parameter

This parameter's main function is to allow you to specify that you want to replace the include file for each class processed, no matter if they already have an include file defined or not. This is useful if you need to do a global replace of include files for some reason. Since this is such a powerful command, you must specifically request this behavior by passing the following text string into the program via this parameter, "ReplaceAll". Anything else results in only those classes that have no include files being processed.

The < cCautious > parameter

This parameter is for those adventurous processors out there. If you do not want to control each replacement of an include file then pass the string "NoMessages" to the program via this parameter. If you are not so adventurous omit this parameter when you call SetIncl( ) and you will be in control of each replacement of each include file in each class definition.

Examples

Replace the include file in all class definitions in all class libraries that do not have include files defined for them with the default include file (..\include\appincl.h for application level classes and ..\include\framincl.h for Codebook level classes) .

```
=SETINCL( )
```

Replace the include file for all class definitions in the project, even if one already exists, and don't warn me about replacements.

```
=SETINCL( "ReplaceAll","NoMessages" )
```

Listing One: SetIncl.PRG

```
*---------------------------------------------------
*-- Example of a cautious setting ...
*--
*--   =SetIncl( "EMPTYONLY", "CAUTIOUS" )
*--           ------ OR ------
*--   =SetIncl()  (defaults to cautious)
*--
*----------------------------------------------------
*-- Example of a ballsy setting ...
*--
*--   =SetIncl( "ReplaceAll", "NoMessages" )
*--
*-- Note: "ReplaceAll" and "NoMessages" are keywords
*--       for the aggressive setting.  Anything
*--       else passed into this program, like the
*--       messages in the cautious setting above,
*--       (1) result in only those classes that
*--       as of now do not have an include file
*--       defined (2) and confirmation messages
*--       being displayed.
*--
*----------------------------------------------------

PARAMETERS tcReplaceAllIncludeFiles , ;
           tcCautious

LOCAL llOK2Replace                        , ;
      lnNumberOfApplicationClassLibraries , ;
      lcFullyQualifiedDefaultIncludeFile  , ;
      lcValidIncludeFile                  , ;
      llCautious                          , ;
      llRetVal

PRIVATE laClassLibNames                     , ;
        lcMessageBoxTitle                   , ;
        llDefaultCautiousValue              , ;
        llDefaultEmptyOnlyValue             , ;
        llBugOut                            , ;
        lcProjectFileAlias

*========================================================*
*==            Application default values            ==*
*========================================================*
#DEFINE   COMMON_INCLUDE   "..\include\framincl.h"
#DEFINE   LOCAL_INCLUDE    "..\include\appincl.h"

DIMENSION laClassLibNames[1]
```

```
laClassLibNames[1] = ""
lcMessageBoxTitle       = "SAVI Include File Utility Message"
llDefaultCautiousValue  = .T.
llDefaultEmptyOnlyValue = .T.


llBugOut                       = .F.
llRetVal                       = .T.
llCautious                     = .T.
lcProjectFileAlias             = ""
llOK2Replace                   = .F.
llReplaceOnlyEmptyIncludeFiles = .T.


*===========================================================================*
*=====                    Main Program Processing                     =====*
*===========================================================================*
CLOSE TABLES
CLOSE DATABASES

llRetVal = llRetVal AND ;
           GetProjectFile( @lcProjectFileAlias )


llRetVal = llRetVal AND ;
           GetClassLibraryNamesFromProject( lcProjectFileAlias )

llRetVal = llRetVal AND ;
           ValidateReplaceAll( tcReplaceAllIncludeFiles         , ;
                               @llReplaceOnlyEmptyIncludeFiles  )

llRetVal = llRetVal AND ;
           ValidateCautionLevel( tcCautious , ;
                                 @llCautious   )

llRetVal = llRetVal AND ;
           MakeIncludeFileReplacements( llReplaceOnlyEmptyIncludeFiles , ;
                                        llCautious                       )

RETURN llRetVal


*===========================================================================*
*=====                Program Processing Functions                    =====*
*===========================================================================*

*============================================
FUNCTION GetProjectFile( tcProjectFileAlias )
*============================================
   LOCAL llRetVal , ;
         lcProjectFile

   lcProjectFile = GETFILE( "PJX", "Project:" )
```

```
IF .NOT. EMPTY( lcProjectFile ) AND ;
   FILE( lcProjectFile )
   llRetVal = .T.
ELSE
   llRetVal = .F.
ENDIF

IF llRetVal
   SELECT 0
   USE( lcProjectFile ) ALIAS project
ENDIF
```

```
   IF USED( 'project')
      tcProjectFileAlias = "project"
   ELSE
      llRetVal = .F.
   ENDIF

   RETURN llRetVal

ENDFUNC

*==========================================================
FUNCTION GetClassLibraryNamesFromProject( tcProjectAlias )
*==========================================================
   LOCAL llRetVal

   SELECT name FROM ( tcProjectAlias ) ;
    WHERE Type = "V"    ;
     INTO CURSOR crsVCX

  IF USED( 'crsVCX') AND RECCOUNT( 'crsVCX' ) > 0
     llRetVal = .T.
  ELSE
     llRetVal = .F.
  ENDIF

  IF USED( tcProjectAlias )
     USE IN ( tcProjectAlias )
  ENDIF

  RETURN llRetVal

ENDFUNC


*=======================================================================
FUNCTION ValidateCautionLevel( tcProvidedCautious, tlValidatedCautious )
*========================================================================
   LOCAL llRetVal, ;
         llParameterIsCharacter

   llRetVal = .T.
   llParameterIsCharacter = ( TYPE( 'tcProvidedCautious' ) == "C" )

   IF llParameterIsCharacter
      tlValidatedCautious = ;
            ( ALLTRIM(UPPER( tcProvidedCautious )) == "NOMESSAGES" )
   ELSE
      tlValidatedCautious = llDefaultCautiousValue
   ENDIF

   RETURN llRetVal
```

```
ENDFUNC
```

```
*================================================================
FUNCTION ValidateReplaceAll( tcProvidedReplaceOnlyEmpty, ;
                                   tlValidReplaceOnlyEmpty )
*================================================================
   LOCAL llRetVal, ;
         llParameterIsCharacter

   llRetVal = .T.
   llParameterIsCharacter = ;
         ( TYPE( 'tcProvidedReplaceOnlyEmpty' ) == "C" )

   IF llParameterIsCharacter
      tlValidReplaceOnEmpty = ;
           ( ALLTRIM(UPPER( tcProvidedReplaceOnlyEmpty ) ) == "REPLACEALL" )
   ELSE
      tlValidReplaceOnlyEmpty = llDefaultEmptyOnlyValue
   ENDIF

   RETURN llRetVal
ENDFUNC


*=============================================================================
FUNCTION MakeIncludeFileReplacements( tlReplaceOnlyEmptyIncludeFiles  , ;
                                      tlCautious                        )
*=============================================================================
   LOCAL lnClassLibraryFile, ;
         lcClassName, ;
         lcIncludeFileName, ;
         lcMessage, ;
         lcDefinedIncludeFile


   *----------------------------------------
   *-- If there are any class library files
   *-- in the current directory
   *----------------------------------------

   SELECT crsVCX
   SCAN

      lcVCX = UPPER(crsVCX.Name)
      WAIT WINDOW "Processing class library ... " + lcVCX NOWAIT

      *----------------------------------------------------------
      *-- All Codebook framework class libraries are named
      *-- LIBS\ as a standard.  Therefore, since this program
      *-- is interested in replacing the include file in Codebook
      *-- classes ONLY, skip the processing of any .VCX files
      *-- that are not stored in the Codebook default locations
      *----------------------------------------------------------
```

```
IF !("LIBS\" $ lcVCX)
   LOOP
ENDIF
lcInclude = IIF( "\COMMON" $ lcVCX , ;
                 COMMON_INCLUDE    , ;
                 LOCAL_INCLUDE       )
SELECT 0
*----------------------------------------------------------------
*-- Open the class library file as if it were a reg'ler ol' table
*----------------------------------------------------------------
USE (lcVCX) ALIAS classlib

IF USED('classlib')

   llAtLeastOneReplacementTookPlace = .F.

   *----------------------------------------------------
   *-- Look for each record in the class library
   *-- that defines a class created by the developer
   *-- Note that this is how FoxPro itself determines
   *-- which records in the class library represent
   *-- classes so they can be displayed when you expand
   *-- the class library in the tree to view the classes.
   *-- The following SCAN/ENDSCAN loop looks for these
   *-- special records ...
   *-----------------------------------------------------

    SCAN FOR "CLASS" == ALLTRIM(UPPER( classlib.reserved1 ) ) AND ;
              EMPTY( parent )

      lcClassName       = UPPER( ALLTRIM( classlib.objname   ) )
      lcIncludeFileName = UPPER( ALLTRIM( classlib.reserved8 ) )
      llOK2Replace      = .F.

      DO CASE
      CASE EMPTY( classlib.reserved8 )
         *------------------------------------------------------------
         *-- It's always OK to replace the include file of an empty
         *-- class
         *------------------------------------------------------------
         lcMessage = "There is no include file for "  + ;
                      CHR(13) + CHR(13)               + ;
                      lcClassName + ".  "             + ;
                      CHR(13) + CHR(13)               + ;
                     "Would you like to change it to" + ;
                      CHR(13) + CHR(13)               + ;
                      lcInclude  + "???"

            llOK2Replace = MessageYNC( lcMessage, tlCautious, @llBugOut )

      CASE .NOT. EMPTY( classlib.reserved8 ) AND ;
           .NOT. tlReplaceOnlyEmptyIncludeFiles
         *------------------------------------------------------------
      *-- If an include file exists and they want to replace it with
      *-- the specified one ... make sure.
      *-------------------------------------------------------------
      lcMessage = lcClassName                               + ;
                  CHR(13) + CHR(13)                         + ;
                 " already has the following include file " + ;
                  CHR(13) + CHR(13)                         + ;
                  lcIncludeFileName + ".  "                 + ;
                  CHR(13) + CHR(13)                         + ;
                 "Do you want to replace it with "          + ;
                  CHR(13) + CHR(13)                         + ;
                  lcInclude + " ??? "

      llOK2Replace = MessageYNC( lcMessage, tlCautious, @llBugout )

   ENDCASE

   IF llBugOut
      EXIT
   ENDIF

   IF llOK2Replace
      REPLACE classlib.reserved8 WITH lcInclude
      llAtLeastOneReplacementTookPlace = .T.
   ENDIF

ENDSCAN

IF USED('classlib')
   USE IN classlib
ENDIF

*-------------------------------------------------------------
*-- If any replacements of include files took place, recompile
*-- the .VCX and it should be ready to go.
*-------------------------------------------------------------
IF llAtLeastOneReplacementTookPlace

   lcVersion = VERSION()

   DO CASE
   CASE "03.00" $ lcVersion
      COMPILE FORM (lcVCX)

   CASE "05.00" $ lcVersion
      COMPILE CLASSLIB (lcVCX)

   OTHERWISE
      *---------------------
      *-- Skip the recompile
      *---------------------
   ENDCASE
ENDIF
```

```
         IF llBugOut
            EXIT
         ENDIF

      ELSE &&- Used( 'ClassLib' )

         LOOP

      ENDIF

      SELECT crsVCX

   ENDSCAN


   IF USED( 'crsVCX' )
      USE IN crsVCX
   ENDIF

   WAIT WINDOW "The Set Include File Utility has completed processing !!!" TIMEOUT
2

   WAIT CLEAR

ENDFUNC
```

```
*==========================================
FUNCTION MessageYN( tcMessage, tlCautious )
*==========================================

LOCAL llRetVal, ;
      lnAnswer

   IF .NOT. tlCautious
      RETURN .T.
   ENDIF

   lnAnswer = MESSAGEBOX( tcMessage, 4, lcMessageBoxTitle )

   DO CASE
   CASE lnAnswer = 6  && Yes
      llRetVal = .T.
   CASE lnAnswer = 7  && No
      llRetVal = .F.
   OTHERWISE          && Unknown (don't take any chances )
      llRetVal = .F.
   ENDCASE
   RETURN llRetVal

ENDFUNC

*=====================================================
FUNCTION MessageYNC( tcMessage, tlOK2Replace, tlBugOut )
*=====================================================

LOCAL llRetVal, ;
      lnAnswer

   tlOK2Replace = .F.
   tlBugOut     = .F.

   lnAnswer = MESSAGEBOX( tcMessage, 3, lcMessageBoxTitle )

   DO CASE
   CASE lnAnswer = 2  && Cancel
      tlBugOut = .T.
   CASE lnAnswer = 6  && Yes
      llRetVal = .T.
   CASE lnAnswer = 7  && No
      llRetVal = .F.
   OTHERWISE          && Unknown (don't take any chances )
      llRetVal = .F.
   ENDCASE

   RETURN llRetVal

ENDFUNC
```

eBizObj Class: How to Make It Work on A Standard Codebook cBizObjMaintForm?

José Constant

Author's Note

This article was written when eBizObj was at its 1.03 revision. It has been greatly enhanced since then and you can safely apply Ed's performance changes to this code also. Every time you find a call to the GetAlias() method, you can replace it with a straight work area save, but don't forget to restore the work area after! Use the new properties like nChildCount instead of Thisform.refBizObj.GetObjectCount()

Introduction

If you plan to use Ed Leafe's ebizobj classes within a standard Codebook cBizObjMaintForm you'll have to add some code. I've also included a few suggestions that make my life easier.

What you should do is probably make a copy of cBizObjMaintForm and then use the class browser to redefine that copy as a subclass of eBizObjForm instead of cBizObjForm. Then, all that is needed is to go through the code and change all the cBizObjForm::Method() calls to eBizObjForm::Method() calls.

I took another approach and eBizObj is now cBizObj in my classes. Following the same design decision, I've enhanced my cBizObjForm with a RefBizObj. So, if you elect the first approach, please bear in mind you'll have to slightly adapt the following code.

Objectives

Well, I want all the components of a cBizObjMaintForm to keep working with any combination of bizness objects onto it. Additionally, I want to be able to use the navigation toolbar, menu and shortcuts when sitting on a child object.

Additions to Codebook base Classes

There are several changes here, just to be sure that the primary bizobj is our oBizObj. I reproduce the full content of this method.

Listing One: cBizObjMaintForm.Requery()

```
LOCAL lnRetVal, ;
  loPage, ;
  laViewParameters[1,2], ;
  lnParameter, ;
  lHaveCriteria, ;
  lnSelectionPage, ;
  lcOldError, ;
  lnChild, ;
  loChild, ;
  lnParent, ;
  loParent, ;
  loSelect, ;
  lnRecCnt

*-----------------------------------------------------------
*-- CHANGE - JCM - October 30, 1996 - 09:34:52
*-- adaptation required for the handling of multiple bizobjs
*-- be sure to have this form set with the primary bizobj as
*-- oBizObj before starting the requery
*-----------------------------------------------------------

FOR lnChild = Thisform.refBizObj.GetObjectCount() TO 1 STEP -1

  loChild = Thisform.refBizObj.Get(lnChild)

  IF TYPE("loChild.cParentBizObj") = "C" AND ;
     EMPTY(loChild.cParentBizObj) AND ;
     loChild.lPrimaryBizObj
     thisform.oBizObj = loChild
  ENDIF

ENDFOR

*------------
*-- Endchange
*------------

lnSelectionPage = 0
*-----------------------------------------------------------------
*-- We test if the pgfBizObj object exists since we may call
*-- Requery() from an Init() of one of the controls on the form.
*-- We test if the Name property is not undefined since the object
*-- will exist but its name has not yet been assigned.
*-----------------------------------------------------------------
IF TYPE("this.pgfBizObj.Name") <> "U"
  lnSelectionPage = this.pgfBizObj.GetPageNumber(PAGE_SELECTIONCRITERIA)
ENDIF

*-----------------------------------------------------------------
*-- If there is no PAGE_SELECTIONCRITERIA page, then just do the
*-- default Requery().
*-----------------------------------------------------------------
IF lnSelectionPage = 0
  RETURN CBizObjForm::Requery()
ENDIF

*-------------------------------------------------------
*-- Get a reference to the Selection Criteria page, and
*-- pick up the view parameters on that page
*-------------------------------------------------------
lnRetVal = FILE_OK
loPage = this.pgfBizObj.Pages(lnSelectionPage)
thisform.GetViewParameters(loPage, @laViewParameters)

*-----------------------------------------------------------------------
*-- Since a parameterized view needs to be able to "see" its parameters,
*-- the trick we use here is to create private variables with the
*-- same names as the view parameters and then call up to the default
*-- Requery() method. These private variables only exist for the
*-- length of time needed to requery the view.
*-----------------------------------------------------------------------
lcOldError = ON('ERROR')

*-----------------------------------------------------------------
*-- An error generated here is most likely a type mismatch error. This
*-- translates to a non-empty non-character field, which means that
*-- we have criteria.
*-----------------------------------------------------------------
ON ERROR lHaveCriteria = .T.
FOR lnParameter = 1 TO ALEN(laViewParameters, 1)
   &laViewParameters[lnParameter, VIEWPARMETER_NAME] = ;
    laViewParameters[lnParameter, VIEWPARMETER_VALUE]
    IF !lHaveCriteria
    lHaveCriteria = !EMPTY(laViewParameters[lnParameter, ;
                           VIEWPARMETER_VALUE])      AND ;
      laViewParameters[lnParameter, VIEWPARMETER_VALUE] <> "%"
  ENDIF
ENDFOR
ON ERROR &lcOldError

*----------------------------------------------------------------------
*-- Set the form property which we can test from elsewhere to determine
*-- if user has entered selection criteria.
*----------------------------------------------------------------------
thisform.lHaveCriteria = lHaveCriteria

*-----------------------------------------------------------------
*-- If no criteria is entered and the business object does not permit
*-- the selection of all records, then inform the user that criteria
*-- must be entered.
IF !lHaveCriteria AND !this.oBizObj.lAllowSelectAll
  =MESSAGEBOX(MUSTENTERCRITERIA_LOC, MB_ICONEXCLAMATION, APPNAME_LOC)
  RETURN FILE_CANCEL
```

```
ENDIF

*------------------------------------
*-- Execute the superclass Requery()
*------------------------------------
lnRetVal = CBizObjForm::Requery()

*------------------------------------------------------------
*-- CHANGE - JCM - October 30, 1996 - 09:34:52
*-- adaptation required for the handling of multiple bizobjs
*-- count the # of records in the father of all bizobjs view
*-- to decide to which page we'll send the user
*------------------------------------------------------------

LOCAL loOldBizObj
loOldBizObj = thisForm.oBizObj
```

```
FOR lnChild = Thisform.refBizObj.GetObjectCount() TO 1 STEP -1
  loChild = Thisform.refBizObj.Get(lnChild)
  IF TYPE("loChild.cParentBizObj") = "C" AND ;
     EMPTY(loChild.cParentBizObj) AND ;
     loChild.lPrimaryBizObj
    thisform.oBizObj = loChild
  ENDIF
ENDFOR

loSelect = CREATEOBJECT("cSelect", this.oBizObj.GetAlias())
lnRecCnt = RECCOUNT()

thisform.oBizObj = (loOldBizObj)

*-----------------------
*-- select the right view
*-----------------------
loSelect = CREATEOBJECT("cSelect", this.oBizObj.GetAlias())

*-----------
*-- Endchange
*-----------

*-------------------------------------------------
*-- If we have data, select the appropriate page.
*-------------------------------------------------
IF !this.IsCursorEmpty()

   *-------------------------------------------------------------
   *-- Leave the list page active if it was active before the Requery(),
   *-- otherwise, select the Data Entry page.
   *-------------------------------------------------------------
   IF this.pgfBizObj.ActivePage <> this.pgfBizObj.GetPageNumber(PAGE_LIST)

      *-------------------------------------------------------------
      *{ Commented at 20:49:51 on July 24, 96
      * this.pgfBizObj.ActivePage = this.pgfBizObj.GetPageNumber(PAGE_DATAENTRY)
      *-- CHANGE - JCM - July 24, 1996 - 20:50:24
      *-- thanks to David Taylor Shannon, 71672,1521
      *-- if more than one record, go to the list page
      *{ Commented at 09:40:01 on October 30, 96
      *  IF _TALLY > 1
      *-------------------------------------------------------------
      IF lnRecCnt > 1
         *} End Commenting 09:40:01 - October 30, 96
         this.pgfBizObj.ActivePage = this.pgfBizObj.GetPageNumber(PAGE_LIST)
      ELSE
         this.pgfBizObj.ActivePage = ;
            this.pgfBizObj.GetPageNumber(PAGE_DATAENTRY)
      ENDIF
         *} End Commenting 20:49:51 - July 24, 96
   ENDIF
ELSE
```

```
  *-- If the Requery() resulted in an empty data set, activate
  *-- the Selection Criteria page
  this.pgfBizObj.ActivePage = lnSelectionPage
ENDIF

RETURN lnRetVal
```

Listing Two: cBizObjMaintForm.pgfBizObj.Page3.Activate()

```
*-------------------------------------------------------
*-- Place this code on top
*-- CHANGE - JCM - October 30, 1996 - 09:34:52
*-- adaptation required for the handling of multiple bizobjs
*-- be sure to have the parent of all BizObjs set
*-------------------------------------------------------
LOCAL lnChild, loChild, lnParent, loParent
FOR lnChild = Thisform.refBizObj.GetObjectCount() TO 1 STEP -1
    loChild = Thisform.refBizObj.Get(lnChild)
    IF TYPE("loChild.cParentBizObj") = "C" AND ;
       EMPTY(loChild.cParentBizObj) AND ;
       loChild.lPrimaryBizObj

       Thisform.oBizObj = loChild
   ENDIF
ENDFOR
```

Listing Three: cToolbar.cRefreshButton.Click()

```
*--------------------------------------------------------
*-- CHANGE - JCM - October 29, 1996 - 15:33:57
*-- adaptation required for the handling of multiple bizobjs
*-- be sure to have the primary BizObj set
*-- because Refresh on a new Parent record would yield an
*-- error if a child is set as primary bizobj
*--------------------------------------------------------
LOCAL lnChild, loChild, lnParent, loParent
FOR lnChild = _Screen.Activeform.refBizObj.GetObjectCount() TO 1 STEP -1
    loChild = _Screen.Activeform.refBizObj.Get(lnChild)
    IF TYPE("loChild.cParentBizObj") = "C" AND ;
       EMPTY(loChild.cParentBizObj) AND ;
       loChild.lPrimaryBizObj
      _Screen.Activeform.oBizObj = loChild
    ENDIF
ENDFOR

*-----------
*-- Endchange
*-----------

RETURN _screen.ActiveForm.Requery()
```

Listing Four: cBizObj.MakePrimaryBizObj()

```
*-----------------------------------------
*-- CHANGE - JCM - April 17, 1997 - 09:16:37
*-- be sure that the toolbars buttons will
*-- refresh appropriately in any case.
*-----------------------------------------

IF TYPE("Thisform.oToolbar") = "O"
   LOCAL loSelect, lnRetVal
   *-----------------------------------------
   *-- be sure to have the right view selected
   *-----------------------------------------

   loSelect = CREATEOBJECT("cSelect", This.GetAlias())

   *------------------------
   *-- calculate our position
   *------------------------
  lnRetVal = IIF(RECCOUNT() > 1, FILE_BOF, ;
               IIF(RECCOUNT() = 0, FILE_NORECORDS, FILE_ONERECORD))
  Thisform.lBOF = .T.
  Thisform.lEOF = (INLIST(lnRetVal, FILE_NORECORDS, FILE_ONERECORD))
  Thisform.oToolbar.Refresh()
ENDIF
```

This change is optional. It simply makes my life easier… [Well, it made my life easier until Ed Leafe came with a better approach in eBizObj V1.04. So, if you use eBizObj V1.04 or higher don't add this code…]

It would be nice if we could also replace the child BizObj foreign key with the parent BizObj primary key value. This is possible with a strong naming convention for the view parameters.

Example: if vp_cRonssId is the view parameter, then cRonssId has to be the foreign key

Listing Five: cBizObj.OnNew()

```
*-----------------------------------------------------------
*-- cBizobj.onnew()
*-- add the following code at the bottom of the OnNew() method
*-----------------------------------------------------------
LOCAL loParent, ;
      lcViewParm, ;
      lcForeignKeyField

IF !EMPTY(this.cParentBizObj) AND ;
   !EMPTY(this.cViewParmName) AND ;
   this.lGetViewParmFromParent
lcViewParm = this.cViewParmName
   PRIVATE &lcViewParm
   loParent = thisForm.GetPArentBizObj(this.cParentBizObj)
   IF TYPE("loParent") = "O"
      &lcViewParm = loParent.GetParentKey()
      *-- contains the value of the view parameter
      lcForeignKeyField = SUBS(this.cViewParmName,4)
      REPLACE (lcForeignKeyField) WITH &lcViewParm
   ENDIF
ENDIF
```

Code that applies to your classes

aYourBizObjMaintForm.pgfBizObj.Page3.grdList

In case you have more than one bizobj on your form, be sure to define the control source of the grid at the grid level also.

PageFrames

If you place your bizobjs onto a pageframe, you'll want to have the toolbar buttons refresh appropriately. For every page that is part of the primaryBizObj, put this code in the activate event:

```
This.Parent.Parent.UIEnable(.T.)
```

For the child BizObjs, you'll have to add this code somewhere:

```
This.MakePrimaryBizObj()
```

Child objects

Grids (1)

In case of a childobj presented in GRID format, you might want to Cancel or Save the changes for all rows in one pass.

Listing Six: ChildObj.Cancel()

```
*------------------------------------------
*--ChildObj.Cancel()
*-- we are handling multiple rows at one time
*------------------------------------------
LPARAMETERS tlAllRows
RETURN cbizObj::Cancel(.T.)

*-----------------
*-- ChildObj.Save()
*-----------------
LPARAMETERS tlAllRows, tlForce
PRIVATE lnRetVal

*--------------------------
*-- save all records at once
*--------------------------

lnRetVal = cBizobj::Save(.T., tlForce)
IF lnRetval = 0
  this.Requery()
ENDIF
RETURN lnRetVal
```

Grids (2)

In case you have a grid on your childBizObj and you want to navigate through it with the navigation buttons, put this code in the AfterRowColChange() method of the grid.

Listing Seven: Grids' AfterRowColChange() method

```
LPARAMETERS nColIndex
=Lockscreen(.T.)

This.Parent.Refresh()


IF TYPE("Thisform.oToolbar") = "O" AND  _TALLY > 1
  LOCAL lnRecNo
  lnRecNo = RECNO()
  LOCATE
  *-----------------------------------------------------------
  *-- If we're still on the same record, we were already on top ...
  *-----------------------------------------------------------
  IF lnRecNo = RECNO()
    Thisform.lBOF = .T.
  ELSE
    thisform.lBOF = .F.
    GO (lnRecNo)
  ENDIF

  GO BOTTOM
  *-----------------------------------------------------------
  *-- If we're still on the same record, we were already at bottom ...
  *-----------------------------------------------------------
  IF lnRecNo = RECNO()
    Thisform.lEOF = .T.
  ELSE
    thisform.lEOF = .F.
    GO (lnRecNo)
  ENDIF

  Thisform.oToolbar.Refresh()
ENDIF

=LockScreen(.F.)
```

Listing Eight: UIEnable method for a childObj

You might want do define the following UIEnable method on the childobj to be sure that it will always refresh.

```
*-------------------
*-- ChildObj.UIEnable
```

```
*--------------------
LPARAMETERS lEnable
LOCAL loSelect, lnRetval
IF lEnable
  This.Requery()
ENDIF
Cbizobj::UIEnable(lEnable)
```

How to avoid flashing effects?

In order to avoid a flashing effect when going from one page to another on a pageframe, especially if this page contains child bizobjs, you may want to add the following code:

Listing Nine: Page.Activate()

```
DODEFAULT()
LockScreen(.T.)
this.YourChildBizObj1.Visible = .T.
LockScreen(.F.)
```

Listing Ten: Page.Deactivate()

```
DODEFAULT()
this.YourChildBizObj1.Visible = .F.
```

© 1997 by José Constant. An engineer, member of the Technology DVL Group and a Microsoft Certified Professional in Visual FoxPro, José offers consulting services at Constant Software Systems in French, Spanish or English. jose@club.innet.be, http://www.club.innet.be/~year2396/

Efficiency and Object Oriented Programming

With the advent of object-orientation in VFP, we have a way of doing things that differs greatly from the old, procedural, Xbase way. We can create objects which automatically perform these actions as part of their Init() and Destroy() methods, and in the process make our code a lot more OOP-y. However, the OOP way is not always the most efficient method to accomplish a task.

Ed Leafe

The Old Way

In Codebook, a great example of this is the way OO events can be used to set and restore work areas. If you can remember way back in the dark days of 2.x, all our procedures began and ended something like this:

```
PROCEDURE Example
  PRIVATE lnSelect
  lnSelect = SELECT(0)  && Save the current work area
  SELECT SomeAlias    && This is the alias we need for this proc

  ... <lotsa code>

  SELECT (lnSelect)   && Restore the original area
RETURN
```

This worked well, as long as you remembered to stick in the final SELECT to set things back the way it was when the procedure was called. But along comes OO, and now we have a foolproof way of doing things.

The New Way

Codebook ships with a handy class named “cSelect”, which does all this for you. In its Init(), it stores the current work area in one of its properties. If you pass it the name of an alias, it will also select that alias. In its Destroy(), it selects the work area it stored in its Init(). And if you scope this object to a local variable, it will automatically be destroyed when you return from the procedure, guaranteeing that you can’t forget to re-select the original work area. Using this object, the example code from above looks like:

```
PROCEDURE Example
  LOCAL loSelect
  loSelect = CREATEOBJECT(“cSelect”, “SomeAlias”)

  ... <lotsa code>

RETURN
```

That's a lot simpler and cleaner. It also is a lot safer, since not everybody (myself included) subscribes to the "one entry-point, one exit-point" style which was emphasized in the days when structured programming was the rage. I sometimes have multiple RETURNs in my code if the nature of the procedure calls for it. But with multiple exit points, you have to remember to re-select your original work area at every one of those exits. Otherwise, your code will give unpredictable results. With the cSelect class, though, it doesn't matter how or where you exit from a method; if it is scoped to a local, that local is released when the method is exited, and the Destroy() of the cSelect() will fire and restore the work area every time.

So, assuming you've read this far, you're probably thinking that you should always use these neat-o objects, since they work better and make your code OOP-ier. But I've just re-written the eBizObj class library to replace all calls to cSelect creation with the old-fashioned method. Why? One simple reason: performance. All this convenience comes at a price.

The Price of OOP

I'm not one to endlessly tweak my code to squeeze out another 0.00001 second from some long routine, but I do like to avoid sluggish apps. Recently I was involved in creating a complex app which contained many, many bizobjs on the main screen. When the user clicked on the 'Next' button, for example, there was a palpable delay before the screen updated with the new record. While all the bizobjs were working properly, the users didn't like the delay at all, and I agreed. I took a look at the code to see what could be optimized. I ran a few benchmark tests with some of my prime suspects, and when I ran one comparing cSelect creation vs. the "old-fashioned" method, the results were very surprising to me. Using cSelect objects took about 4 times as long as manually saving and resetting the work areas!

The Solution

Now since the code for eBizObj isn't changed by developers, there was no danger of inadvertently leaving out the work-area re-selection call. Once the change was made and tested, the class wouldn't vary from project to project, so it is just as safe. I replaced all the calls to cSelect with the old construct, and the app's responsiveness picked up noticeably. This stands to reason, since every level of every bizobj calls cSelect before "doing its thing", so in a complex form with over 2 dozen inter-related bizobjs, the difference was significant.

So now the code for eBizObj looked like:

```
PROCEDURE Example
  LOCAL lnSelect
  lnSelect = SELECT(0)
  SELECT (This.GetAlias())

  ... <lotsa code>

  SELECT (lnSelect)
RETURN
```

One other thing which caught my eye was the call to This.GetAlias(). I'm sure someone will be able to come up with an example to refute this, but in the many bizobjs I've created in my apps, I don’t recall a single one whose primary alias (i.e., that which is returned by This.GetAlias()) has ever changed during the lifetime of any particular instance. Once the object is created, the value returned by GetAlias() is constant.

Why is that important? Well, a few months ago I was working with Menachem Bazian of Flash on a project where a huge number of transactions had to be processed, and every millisecond counted. He noticed that calling out to a method was a lot slower than keeping the code in the same method. The example he used looked like this:

```
[First test; using straight code]
LOCAL lcText
lcText = “Hello”
? lcText
RETURN

[Second test; using function call]
LOCAL lcText
lcText = “Hello”
=PrintIt(lcText)
RETURN

PROCEDURE Printit
  LPARAMETERS tcText
  ?tcText
RETURN
```

Now looking at those two variations, it is obvious that they do the exact same thing. But when they were run in a benchmarking program, the latter took about 5 times as long to complete! I recently re-ran this example in VFP 5.0 and found similar results.

Given this, I decided to replace the repeated calls to This.GetAlias() with a reference to a newly-added property of eBizObj called, naturally enough, ‘cAlias’. This property is set once at Init() time by a single call to This.GetAlias(); after that, all methods which need to now the alias to work with simple reference the property rather than call a method. The result of these two changes is a noticeable increase in performance in the complex application mentioned above. It can still use some tweaking, of course, but at least I know that eBizObj is running much more efficiently.

So the final version of the code in eBizObj is:

```
PROCEDURE Example
  LOCAL lnSelect
  lnSelect = SELECT(0)
  SELECT (This.cAlias)

  ... <lotsa code>

  SELECT (lnSelect)
RETURN
```

The Moral

So if there is a moral to this story, it is not to accept theories blindly without testing them practically. The cSelect class is a great example of a solid, practical, smart utility class which embodies all that is powerful and convenient in OOP, but by running a few tests we find out that VFP hasn't yet reached the point where its objects can perform as efficiently as we may like.

You can always get the latest version of the eBizObj class library from my website, at <http://www.leafe.com/dl.html>.