THE CODEBOOK NEWS

August 1997

Volume 1, Issue 2

TABLE OF CONTENTS

Editor's Comments

Charles T. Blankenship

This second issue of The Codebook News owes many thanks to José Constant and his security article. Ensure you pay particular attention to the concept of the security sticker, this is an extremely interesting use of a label object for controlling the behavior of a form. Notice that it is visible at design time but invisible during run time. Don't forget that included with this issue are the class libraries provided by José to accompany his article.

The article Programming in the Problem Domain was inspired by Steve McConnell's book Code Complete. In his book, Steve accurately identifies that most programmers do not make adequate use of abstraction in their coding which causes their professional lives to be much more difficult than it needs to be. The problem I had with his assessment of "many programmers" was that I fell into that category. I hate one and only one thing more than having someone tell me I am doing something terribly wrong and that is when they are absolutely right about it. Since I am not in the business of making money for doing things wrong I did some research to correct the problem; the results are in this article. To determine if you too fall into Steve's classification of "many programmers" go to one of your push buttons (the place where much action, read "business processing", takes place) and examine what resides there. If you see line after line of VFP code whose sole purpose is to accomplish some business task defined by the user then Steve has you in his net as well. Let me know if this article affected you in a positive (or negative) way.

There is one thing I must point out about the concepts presented in the article Programming in the Problem Domain, it is the last step in the development process. As you have heard many times before, object oriented development requires much more analysis and design than functional programming does. Before you can implement an application using these concepts you must have a complete picture of what the application must do. In order to accomplish this you must perform adequate requirements gathering, analysis and design. This article deals with implementation. In many articles to follow I will illustrate one method (initially proposed by Scott Ambler of Software Development Magazine) to use when gathering and analyzing requirements, designing and finally implementing an OO system.

In order to illustrate the preliminary steps to implementation I plan to use the User Communications module I am developing as an example. This is a modular enhancement to Codebook that provides the user with the capability of communicating with the developer for the purpose of identifying/requesting the fixes of bugs (maintenance requests), general comments, or functionality enhancements (enhancement requests). This module collects data in .DBF and sends them via eMail to the developer who then takes appropriate action. I plan to incorporate many of the concepts provided by Any Neil in his recent article "Improve Your Testing Processes" into this on-line form of communication between developer and client so that this module can be used while beta testing too. As each issue comes out I encourage you to participate in the design process (either in 3rd Party Products on the FoxUser Forum or Ed's server when it becomes functional). This is the first jointly developed framework enhancement to be offered to the Codebook community. Your comments and design needs are critical to its success.

Don't forget that Mr. Leafe is currently working hard on a forum (on his server which is also free) that will be dedicated to CB development. Now, instead of fighting through a list of other third party product information we will have a place to do just Codebook work. Hopefully, this is also where Codebook developers can congregate and design new and more powerful extensions to the Codebook framework. The purpose of this is to increase the number of problem domains for which the Codebook framework can be used to develop high quality applications (at an alarmingly fast rate).

Finally, as you all know, I am a pure Codebook fanatic and am reluctant to look for alternatives to a solution that works so well. ... even a Codebook based one. However, I just had the incredible opportunity to use Mike and Toni Feltman's Visual Fox Express and was not only impressed but truly astounded by how much it decreased by development time. This product (as many of you already know) deserves complete investigation ... and it will receive plenty of it here since it is based on Codebook. So, if there are any VFE fanatics out there who want to get their discoveries known to a reader base of approx. 75 developers (and growing) submit articles that document your discoveries and experiences with VFE to The Codebook News for publication.

Once again thanks must go to the other editors of this magazine, Ed Leafe, Frank Cazabon, José Constant and Kevin Emmrich for catching the discrepancies that I don't. Your contributions are greatly appreciated by all.

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*

Programming In The Problem Domain

"Many programmers never work above this level of abstraction (Level 1: High-level -- Language Structures), which makes their lives much harder than they need to be" Steve McConnell, Code Complete.

Charles T.Blankenship

Introduction

In his book, Code Complete (Microsoft Press, ISBN 1-55615-484-4, $35.00), Chapter 32: Themes In Software Craftsmanship, Section 32.5: Program In Terms of the Problem Domain, Steve McConnell illustrates how the concept and proper application of abstraction makes a programmer's life much easier. This article explores one possible method of programming in the problem domain using the Codebook framework.

Understanding The Theoretical Levels Of Abstraction

The first step in the process of learning how to program in the problem domain involves understanding Mr. McConnell's concept of abstraction. It is very simple, like most good ideas are. To achieve abstraction, a name is used to accurately describe the function of a group of Visual FoxPro commands. That descriptive name, derived from the terminology used in the problem domain, is the key to gradually introducing the maintenance programmer to the complexity of the custom application.

This form of abstraction is not foreign to anyone who develops applications in a 4GL language like Visual FoxPro. Every time a programmer creates an object in the computer's memory using the CREATEOBJ( ) command this concept is put to use. In this case, however, a Microsoft programmer is isolating the application developer from the complexity of *really* creating an object in a computer's memory by wrapping up several hundred lines of C++ code (probably) with a descriptive name like CREATEOBJ. Programming in the problem domain is simply an extension of this concept to a higher level of abstraction. To begin programming in the problem domain, an application programmer has only to wrap up Visual FoxPro commands that accomplish a specific business process with a descriptive name derived from the terminology of the business' problem domain.

A Metaphor

A good metaphor for this type of abstraction is a book. At the highest level of organization, a book usually has a table of contents that identifies and describes each chapter. The next level of organization, the chapter, frequently has *its* own table of contents that describes each of the sections in the chapter. Finally, the lowest level of organization, the section, contains the actual words that describe the section topic in detail. The goal of using abstraction in programming is to make the program as well organized as a professionally published book.

The Challenge

Four levels of abstraction are required when designing a software application in order to experience the benefits of programming in the problem domain. Code Complete gives a general description of each of these levels. The main problem to overcome is finding programming constructs in VFP/Codebook that correspond to each of the four levels of abstraction defined in Code Complete.

The rest of this article describes each level of abstraction presented in Code Complete, a proposed VFP/Codebook construct to use to implement each level of abstraction, rules to follow when creating level one

through level four abstractions, and finally, an example that illustrates the benefits of programming in the problem domain using code that should be very familiar to you.

Level One: High-Level Language Structures

High-level language structures are the language's primitive data types, control structures, and so on. Visual FoxPro commands like CREATEOBJ( ), ISNULL( ), USED( ), etc., are all Level One abstractions. These commands and functions remove the application programmer from the true complexity of performing these actions in the machine by one level (some would argue more than one level) by wrapping hundreds or thousands of lines of C/C++/Assembler code in a very descriptive name that the programmer can easily remember from application to application.

Application developers take these commands and string them together in unique and clever ways to solve problems for businesses. The unfortunate fact of the matter is that most programs are written in nothing but these high level language structures. Programs written like this have nothing but line after line of Visual FoxPro code performing the processes that make up the application. These programs do nothing to isolate the maintenance programmer from the complexity of the application's processes. As a result, applications developed in this manner seldom experience the benefits that abstraction can bring, ease of maintenance, reuse and flexibility. As per Steve McConnell, "Many programmers never work above this level of abstraction, which makes their lives much harder than they need to be."

Level Two: Computer Science Structures

This level of abstraction is composed of structures that are "slightly higher-level ... than those provided by the language itself." Examples include "stacks, cues, linked lists, trees, indexed files, sequential files, sort algorithms, search algorithms, etc." According to Mr. McConnell, "this level represents an improvement in abstraction over high-level-language structures but still contains too much detail to be able to win the battle against complexity."

The problem is determining which structure in VFP/Codebook is best used to implement this level of abstraction. The ideal VFP construct must first allow the programmer to string FoxPro commands together to accomplish one specific task. Secondly it must provide a vehicle to label that functionality using a descriptive name. Private methods of Codebook business objects are an excellent fit. These methods provide a convenient place to group, protect and descriptively name processing logic for business objects. The primary purpose of these methods is to create the functional building blocks upon which to build the higher level process of the business object. The names of some hypothetical private methods follow:

CreatePrintingCursor() - creates a duplicate of the primary table to store records to be printed
DestroyPrintingCursor() - destroys the temporary printing cursor
CreatePrintingCursorIndex() - ensures the printing cursor is properly ordered

Notice that the names of these methods still contain computer jargon like "Cursor" and "Index". This is acceptable because these methods are performing "low level" operations ... well, low-level as far as VFP applications are concerned. The function of this level of abstraction is to encapsulate the complexity required to manipulate files, arrays, and cursors, indexes, massaging data, etc, which are used to solve problems in the problem domain. Therefore, the names used to describe these processes are necessarily technical in nature.

Based on this information, the first rules to abide by to begin programming in the problem domain follow:

Rule 1: The primitive functions of the business object, procedural building blocks so to speak, have their processing logic encapsulated in protected methods of the object's class definition.

Rule 2: The names of the private methods must adequately describe the function being performed. The name of the protected methods can contain technical jargon since the operations being accomplished work directly with computer science structures.

Level Three: Low-level Problem Domain Terms

This description was so well done in Code Complete that I simply have nothing else to offer except to transcribe it for you:

> "At this level, you have the primitives (private methods of the business objects) you need in order to work in terms of the problem domain. It's a glue layer between the computer-science structures below and the high-level problem-domain code above. To write code at this level, you need to figure out the vocabulary of the problem area and the building blocks you need in order to work with the problem the program solves. Routines at this level provide the vocabulary and the building blocks. At this level, the routines might be too primitive to be used to solve the problem directly, but they provide an Erector set that higher-level routines can use to build a solution to the problem."

```
If it isn't obvious yet, you should run, not walk, to the Microsoft Press on the I*Net and
buy this book.  This article only deals with one section.  There are thirty three chapters
in this book packed with equally wonderful topics ... go ... now ... I'll wait to finish
this article when you get back ...
```

As with the previous level, the first task is to determine which VFP/Codebook construct can house Level Three abstractions. Since the private methods of the business objects hold level two abstractions, the only logical choice to hold level three abstractions is the public methods of the business objects.

The second task is to determine how to construct level three abstractions. Code Complete provided some clues when it said you "have the primitives you need to work in the terms of the problem domain" present in the private methods of the business objects. Therefore, one solution, is to use the private methods of the business object to program the functionality of the business object's public methods. This is a logical progression of Code Complete's abstraction theory. When programming with Visual FoxPro every developer must use the commands provided by Visual FoxPro, Level One Abstractions, to create the business object's private methods, Level Two Abstractions. It is not illogical to assume that constructs used to create public methods Level Three Abstractions, use the building blocks provided by the immediately preceding level of abstraction, the private methods. Therefore, programming in the problem domain requires an additional rule:

Rule 3: Private methods of the entity objects are the building blocks with which the higher level functionality of the business object is created. The only VFP commands contained at this level are control structures such as DO CASE, DO WHILE, IF-ELSE-ENDIF, etc.

It is very important to remember that raw VFP language commands are never used at the business object public level. [There is one possible exception to this rule.]

The next step in discovering the attributes of level three abstractions involved formulating guidelines for naming them. Once again, Code Complete provided the appropriate hints with the statement "To write code at this level, you need to figure out the vocabulary of the problem area ..." Therefore, at this level of abstraction, the computer jargon completely disappears from the method names and is supplanted by the terminology used every day in the business for which this program is written.

Rule 4: The names of the public methods of the entity object contain only the terminology of the problem domain. No computer jargon ever appears at this level.

Some examples of methods named in this fashion are:

CreateNewSlideShow( )
ExportWeaponCharts( )
PrintWhatIfAnalysisReport( )

The absolute requirement for the name of each public method is that the business people for whom this program is written must understand exactly what that method does by looking at its name. They may not understand the functionality of the more cryptic "primitive" processes that actually accomplish the task, but they certainly know what printing a What If Analysis Report means to them.

Rule 5: The public methods of entity objects never use methods from other objects to accomplish their tasks. Only the private methods of the entity object are used to accomplish the business oriented task of the business object.

The purpose of Rule 3 and Rule 5 is to ensure that business objects are 100% self-contained, self-sufficient and that the coupling between business objects is reduced to zero.

The question that is probably springing to mind is, "If business objects are not allowed to directly collaborate with one another, how does any complex tasks get accomplished?" An excellent question, the answer lies in the fourth and final level of abstraction. High-level Problem Domain Terms.

Level Four: High-level Problem-Domain Terms

The characteristics of high level problem domain terms, as per Code Complete, are as follows:

1. "the code should be somewhat readable by someone who is not a computer-science whiz"
2. "Code at this level won't depend much on the specific features of your programming language because you will have built your own set of tools to work with the problem."

Just as the public methods of the business objects relied upon the private methods (tools) to complete their functionality; the business object public methods themselves are used as building blocks for the next higher layer of abstraction.

The problem is that we have run out of Codebook constructs. Therefore, the construction of a custom object is in order. This new object must be capable of mediating the public properties and methods of the business objects for the sole purpose of accomplishing a high level business ***event***, oooo, good name. The newly named *event object* is responsible for managing high level business transactions.

Rule 6: Event objects use the public methods of the business objects to accomplish their assigned task. They are responsible for coordinating multiple messages to different business objects in a specific order and interpreting the results for the sole purpose of completing a high level business event.

Several very important characteristics of the event object bear mentioning. The first characteristic of this level of abstraction is that the event method is somewhat readable by the business people commissioning the writing of this software. The reason for this is that the code of an event object is composed mainly of references to the public methods of the participating business objects, and the names of those public methods are abstractions which common business people can understand. The second characteristic is that the analysis

and design that went into the creation of the software is embodied in the software itself at this level using this technique. The event method code reads like a requirements specification. This enables programmers who must maintain the software to understand it much more quickly. The quicker the code can be understood, the quicker the programmer can complete the maintenance. The quicker the maintenance is completed the cheaper the software is to own. As soon as you convince your clients that your software is much cheaper to maintain than the other guy's/or gal's, the quicker you get the programming job in the first place.

Most clients understand how expensive it is to commission the development of custom software. All clients who have ever had to maintain custom software realize that the cost of the development phase is much less than the total cost of the maintenance phase over the life of the software. This is a very strong selling point for software developed in this fashion.

Programming In The Problem Domain: Visual Representation (Figure 1)

Programming in the problem domain relies heavily upon the concept of the "building block". Programmers use the development language to create primitive "building blocks" of functionality. The "building blocks" created in each previous level act as the tools/raw material required to build the functionality of each successive layer of abstraction.

Basically, this concept is identical to the one used by companies that develop languages like FoxPro. Microsoft used C/C++/Assembler to create commands and functions that enable FoxPro programmers to easily accomplish tasks with a computer. The concept of programming in the problem domain should be approached as if the application developer is developing a custom computer programming language for the business. This programming language allows the business to use the computer to solve business problems using the terminology common to their business environment.

Comptuer Science Structures

Notice that Visual FoxPro (a high level language structure) is the tool used to create the functionality of the protected methods of the business objects (computer science structures). These methods manage computer science structures like arrays, tables, cursors, indexes, link lists, etc. Basically, any constructs made available by the FoxPro language that help a programmer automate a business process has the code that manages that structure created here. This ensures the complexity of managing that structure is encapsulated within this level. This is represented on the illustration by a line connecting the protected method with the Visual FoxPro Language.

The benefits received at this level of abstraction are increased quality and maintainability. These methods have very specific responsibilities. As a result, it may only take five or ten lines of code to accomplish a task. When a method is this simple, it is easy to determine if there is a logical flaw in the processing. A simple method is also easier to test. If testing is easy, the developer is more like to do it. Since the entire application is built using small, thoroughly tested methods, the number of bugs that reach the production environment are very small in number as well as severity.

The next benefit is that code is easily maintained. Methods developed in this manner are so small and simple that a relatively novice developer can easily understand the process being automated. The easier the process is to understand, the easier it is to fix. If it is easy to fix it is cheaper to fix ... for two reasons. First, the level of expertise required by the maintenance programmer is significantly reduced. Junior level programmers can perform this task with relative ease. Second, the time to fix a simple process, even by a junior level programmer is usually small. Less time at a smaller hourly rate means a great savings during the maintenance phase of the system life cycle.

Low-level business processes

The protected methods of the business objects provide the resources to create the functionality of the public methods (low-level problem domain terms). At this level, the programmer has the first opportunity to solve a business related problem with a primitive tool created at the previous level of abstraction. This is represented on the illustration when PubMethodOne() uses PriMethodOne(), PriMethodThree() and PriMethodFour() to accomplish its task.

The primary benefit of programming in the problem domain at this level of abstraction is increased reuse. The private methods of the business object provide the functional foundation upon which the low level business processing of the business objects rest. Since the business object is a very cohesive unit, it is highly probable that a process used to automate one low level business process can be used to help automate another. Therefore, this design promotes intraobject reuse, this is where a private method is used by two or more public methods of the same business object.

The second benefit realized by a program designed in this manner is that the true functionality of the business object is protected within the business object itself, within its protected methods. This effectively hides the implementation of the object safely within the object where it cannot be used inappropriately.

The third benefit is an increase in maintenance efficiency. Since common procedures are programmed in one place, fixes only have to be made once. If a new implementation of a private method increases performance, the performance increase benefits all public methods that use it. In effect, this allows a programmer to normalize code in the same manner the Database Administrator normalizes databases and effectively conveys the same benefits.

The fourth benefit is increased maintainability. Since the use of high level language structures is kept to a minimum (only decision structures should be used at this level) and since the computer science structures of the previous level are descriptively and clearly named, the code at this level of abstraction reads like a very technical requirements specification. This effectively helps capture the design decisions in the code itself, not in some CASE tool that is separate and distinct from the application being maintained.

High Level Business Processes

The characteristic common to all business objects is that they only contain methods whose processing is directly responsible for manipulating the data stored in the table for which they are responsible, i.e., there is a one to one correlation between the business object and the table it encapsulates.

Very few high level business processes are simple enough to be completed with one table in a company's database. Usually, a high level business process like recording a sale involves the cooperation of several company tables, among them Customer, Invoice, Purchase (Invoice Line Item), Salesman, Inventory, etc. Processes in each of the business objects associated with these tables must be called in a specific order with a specific branching logic in order to accomplish the task. This is the responsibility of the event object. For example, methods in the CustomerBizObj, InvoiceBizObj, PurchaseBizObj, SalesmanBizObj and InventoryBizObj must be brought together in such a way to successfully accomplish the sale. If an error occurs, the event object is responsible for cleaning up after itself.

The first benefit realized from this level of abstraction is increased maintainability. The "code" at the fourth level of abstraction, the high level business process, is somewhat understandable by the

businessmen/women themselves. This is because the "code" is composed of low-level business processes the names of which are composed of terminology that business person uses in their business lives. If a business person can understand the "code" that comprises this method it is even easier for a maintenance programmer to understand it.

The second benefit is that the substance of the application's analysis phase is captured in the code itself. It is not only present in some CASE tool separate from the code. The result is practically self documenting code. Most programmers do not like to document extensively. This methodology provides a way to document processing by using highly descriptive names for the methods used to accomplish the task.

An Example of Programming in the Problem Domain

A picture is worth a thousand words and so is an example. The following code was taken from Codebook itself, the cDataEnvironment::Init() method to be exact. Notice that the documentation provided by Flash was removed for a very specific purpose.

Take a moment to examine Listing One. Use a stopwatch and measure how long it takes to understand what is being accomplished. It took me about 10 minutes (600 seconds) to become comfortable with it.

Take another moment to examine listing two. Start a stopwatch and perform the same test. It took me about 10 seconds to reach the same comfort level. This is a 6000% increase in efficiency. The complete functionality required to create a data environment can be now easily understood, at a high level. Any desire to determine exactly how the default database is valid is available in a method at a lower level of abstraction, IsValidDefaultDatabase().

Finally, examine listing three. This is an example of a Level Two Abstraction where Visual FoxPro commands (Level One Abstractions) are used to determine whether or not the default database name is valid. Examine this method closely. How long does it take to become comfortable with this processing? The time can be easily measured in seconds, even for novice (cheap to employ) developers.

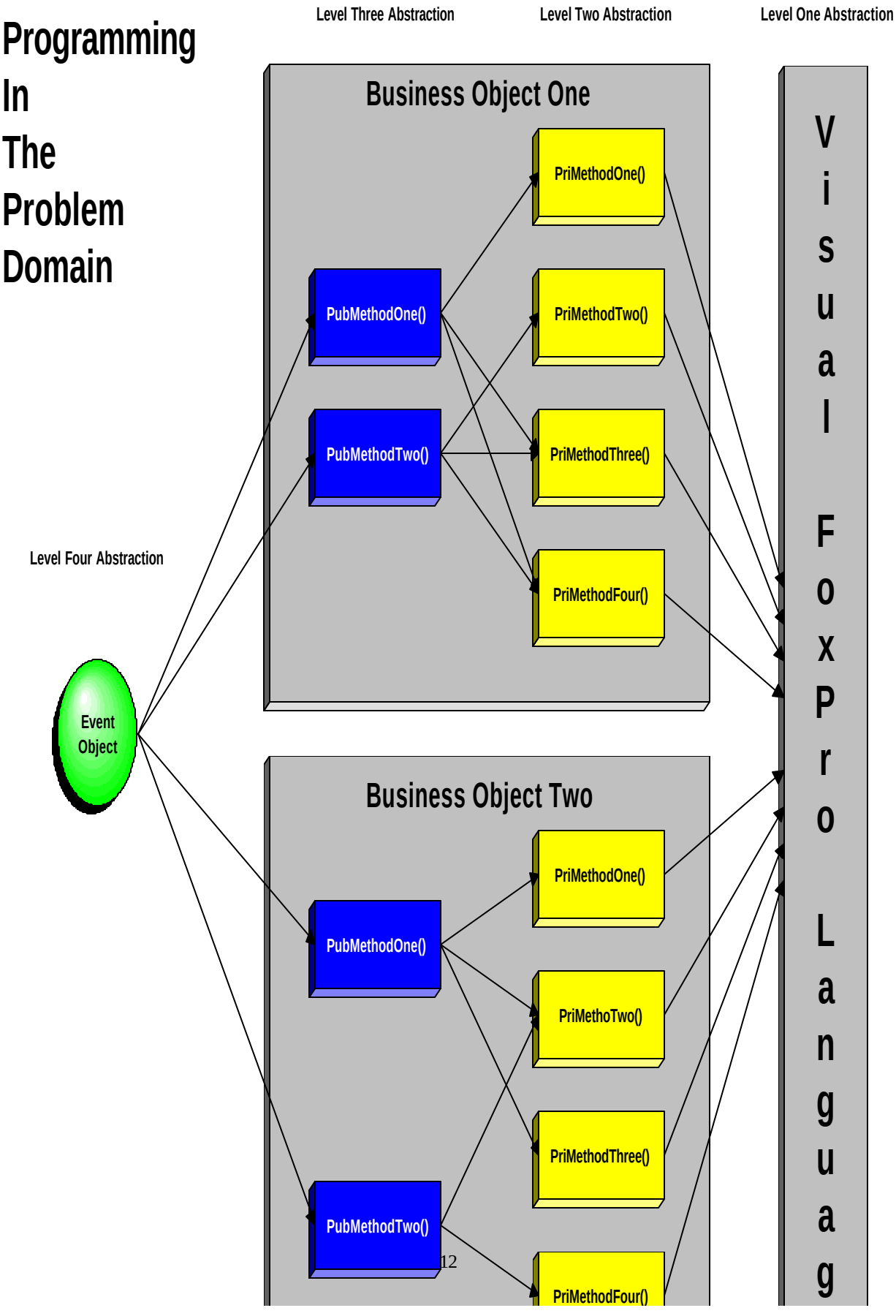
Conclusion

Programming in the problem domain has allowed me to greatly increase my code's maintainability and flexibility while simultaneously decreasing the cost of ownership. The only condition guaranteed in a business environment is that it changes, frequently and sometimes drastically. The companies in these environments that rely on computers to quickly and accurately automate their critical processing need those critical applications to be as flexible to change as the operating environment is susceptible to it. Those changes must be made quickly, accurately and economically. These requirements are difficult to meet when raw VFP code is imbedded in hundreds of interface objects throughout the entire application. It is only possible when the design of the software provides application developer's with the building blocks necessary to create the new functionality required to meet the changing business environment.

Programming in the problem domain in the software industry is analogous to the use of standard, dimensioned raw materials in the construction industry. Standardized lumber, windows, shingles, et cetera, allow a skilled carpenter to quickly build any house a dreamy-eyed homeowner can get the plans for. Correspondingly, the class definitions of Codebook coupled with the concepts of programming in the problem domain allow Codebook developers to first create the building blocks necessary to build a software solution for their clients. Secondly, those building blocks can be used to create a well designed, highly flexible, easily maintained software solution for their clients.

This new implementation of cDataEnvironment is available for download at *www.savvysolutions.com*. Follow the Framework Enhancement link from the homepage.

Figure 1: Programming In The Problem Domain, Visual Representation



Listing One: Original cDataEnvironment::Init() code from CDATAENV.PRG

```
FUNCTION Init()
   ********************************************************
   *   Copyright:  Flash Creative Managememt, Inc, 1995
   ********************************************************
   IF IsAbstract(this.Class, "CDataEnvironment")
      RETURN .F.
   ENDIF

   this.cDefaultDatabase = this.GetDefaultDatabase()

   IF EMPTY(this.cDefaultDatabase) OR !FILE(this.cDefaultDatabase)
     =ErrorMsg(CANNOTFINDDATABASE_LOC + " " + this.cDefaultDatabaseName)
     RETURN .F.
   ENDIF

   LOCAL lnCursor, lnRelation
   this.LoadCursors()
   IF TYPE("this.aCursors[1]") == "C"
      FOR lnCursor = 1 TO ALEN(this.aCursors, 1)
         this.AddObject("o" + this.aCursors[lnCursor], this.aCursors[lnCursor])
      ENDFOR
   ENDIF

   IF TYPE("this.aCursors[1]") == "C"
      this.InitialSelectedAlias = this.aCursors[this.nInitialSelectedAlias]
   ENDIF

   this.LoadRelations()
   IF TYPE("this.aRelations[1]") == "C"
      FOR lnRelation = 1 TO ALEN(this.aRelations, 1)
         this.AddObject("o" + this.aRelations[lnRelation], ;
                               this.aRelations[lnRelation])
      ENDFOR
   ENDIF

   IF this.AutoOpenTables
      this.OpenTables()
   ELSE
      IF TYPE("this.aCursors[1]") == "C"
         lcCursorParentClass = ;
            EVAL('this.o'+this.InitialSelectedAlias+'.ParentClass')
         lcCursorParentClass = UPPER( lcCursorParentClass )
         IF lcCursorParentClass != 'CFREETABLECURSOR'
             SET DATABASE TO (EVAL("this.o" + this.InitialSelectedAlias + ".Database"))
          ENDIF
          SELECT this.aCursors[1]
       ENDIF
    ENDIF

  ENDFUNC
```

Listing Two: The Same cDataEnvironment::Init() code re-engineered using the principles of Programming in the Problem Domain. This is an example of a Level Three Abstraction.

```
FUNCTION Init()
LOCAL llRetVal, ;
      lnCursor, ;
      lnRelation

   llRetVal = .NOT. IsAbstract(this.Class, "CDataEnvironment")
   IF llRetVal
      this.cDefaultDatabase = this.GetDefaultDatabase()
   ENDIF
   llRetVal = llRetVal AND THIS.IsValidDefaultDatabase()
   IF llRetVal
      THIS.LoadCursors()
      THIS.CreateCursors()
      THIS.SetInitiallySelectedAlias()
      THIS.LoadRelations()
      THIS.CreateRelations()
      IF THIS.AutoOpenTables
         THIS.OpenTables()
         THIS.SetDefaultDatabase(.F.)
      ELSE
         THIS.SetDefaultDatabase(.T.)
      ENDIF &&-THIS.AutoOpenTables()
   ENDIF &&-llRetVal

ENDFUNC
```

Listing Three: Example of a level two abstraction. FoxPro Commands (Level One Abstractions) are used to accomplish a very specific task.

```
FUNCTION IsValidDefaultDatabase()
   LOCAL llRetVal
   llRetVal = .T.

   IF EMPTY(this.cDefaultDatabase) OR !FILE(this.cDefaultDatabase)
      =ErrorMsg(CANNOTFINDDATABASE_LOC + " " + this.cDefaultDatabaseName)
      llRetVal = .F.
   ENDIF

   RETURN llRetVal

ENDFUNC
```

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*

Add Password Security to Your CodeBook Applications

by José Constant

Introduction

The Visual FoxPro Codebook by Yair Alan Griver (published by Sybex) provides the basic hooks to which you can safely tie your own security system. The one described hereafter is largely inspired by the Tastrade example provided with VFP 3 & 5. So, if you feel this text is too concise, you might always want to take a look at Tastrade, it should not hurt ...

Thanks to Yag, Christian Desbourse, David M. Tacke, Dejan Jocic, Bill Gatewood, Bill Anderson, Quoc-Buu Truong and Frank Cazabon for revising this document and enhancing it. All the suggested changes / modifications have been integrated in this version of the document.

Specification

The system must be easy to add to existing apps. Ideally, most of its intelligence should be stored in a common library that we could use throughout all our applications.
Limit user categories to 4 types : developer, system administrator, system user and guest. You could add more, we just kept it to 4 for clarity. You'll find in the following chart their respective rights.

	Developer	System admin.	System user	Guest
No need to provide password	X[1]			
Define System Users and password	X	X		
Additional menu options (depend on application)	X	X		
System preferences	X	X		
Login	X	X	X	X
Change password	X	X	X	X

- Menus : instead of dimming certain menu options for common users, we want them to never appear. It adds clarity to the application and avoids user frustration since he cannot ask himself *" Why do I never gain access to this option ? "*
- The same reasoning applies to the toolbar buttons;
- Forms : restrict user access based on their level. Ideally, this should be as easy as dropping a label onto the form. We can decide if we let our users Edit, Delete or Add records, or any combination of the three.
-

As you can see, this is a very basic security scheme. You might want to enhance it later with a few more " goodies " like :

- password encryption : this might be achieved easily with an encrypting / decrypting library like CIPHER.FLL (available on CompuServe, FOXUSER, LIB9)
- remove the user from the user's list after 3 unsuccessful logon attempts
- force password renewal every x days

[1] Only in debug mode, otherwise has to enter his password as any other user

- close user accounts at a given date
- keep an audit trail of user logins
- etc ... There's virtually no limit.
-

Let's see now how this is achieved.

Strings.H and Strings.DBF

These two files are used throughout all CodeBook 3 applications. Any addition you make here will be reflected in all your CDBK3 projects.

Strings.H

First, you'll add the following #DEFINE to your strings.H

```
*-- CHANGE - JCM - August 04, 1996 - 11:30:22
*-- user level description
#DEFINE USER_APPDEV_LOC           GetString("USER_APPDEV_LOC")
#DEFINE USER_OPSMGR_LOC           GetSTring("USER_OPSMGR_LOC")
#DEFINE USER_COMMON_LOC           GetString("USER_COMMON_LOC")
#DEFINE USER_INVITE_LOC           GetString("USER_INVITE_LOC")

*-- CHANGE - JCM - August 05, 1996 - 21:30:02
*-- password things
#DEFINE BADPASSWORD_LOC    GetString("BADPASSWORD_LOC")
#DEFINE NOPSDWENTERED_LOC         GetString("NOPSWDENTERED_LOC")
#DEFINE PASSWORDEMPTY_LOC         GetString("PASSWORDEMPTY_LOC")
#DEFINE PSWDNOTCNFRM_LOC          GetString("PSWDNOTCNFRM_LOC")
```

Strings.DBF

This table contains the translation for all the framework strings. You'll have to add a line for every #DEFINE. By doing so, we can share the same code, should we be in the US or France or Spain. We provide here the French strings ...

Key	String
USER_APPDEV_LOC	Développeur
USER_OPSMGR_LOC	Administrateur du système
USER_COMMON_LOC	Utilisateur du système
USER_INVITE_LOC	Invité du système
BADPASSWORD_LOC	Le mot de passe est incorrect!
NOPSDWENTERED_LOC	Vous n'avez pas encore saisi l'ancien mot de passe! Voulez-vous continuer?
PASSWORDEMPTY_LOC	Le nouveau mot de passe ne peut être vide!
PSWDNOTCNFRM_LOC	Le nouveau mot de passe ne peut pas être confirmé. Veuillez essayer à nouveau.

New Tables

Introduction

These tables will be added to the application Database Container.

For those who plan to use this security module as a standard feature, they might as well add these 2 tables to their generic project, so that they will be created with every new project.

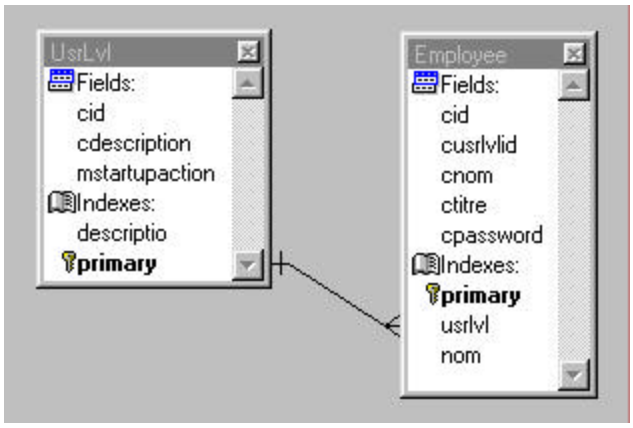


Figure 1 : UsrLvl and Employee Tables in relation

Table UsrLvl

As you might have guessed already, it defines our 4 different user levels. There is a cid as our primary key, – a standard CodeBook practice, – a cDescription field filled with the exact same strings as in Strings.DBF (for our first 4 #DEFINE – see above chart) and an mStartupaction that will be used by the application object GetStartupAction() method to initiate an action at startup.

As an example, you might want to put the System Administrator in the Administration menu pad, so you'd store KEYBOARD "{ALT+F}" . The action is just a FoxPro command stored as character.

Beware of mStartupAction! If you store "goApp.oMenu.oAdministrationPad.oPopup.Show() " in it, to be more OOP and avoid KEYBOARD commands, the system hangs when you want to exit by selecting the File Exit menu option. If you try to close again, App comes up with error "Object goApp is not found". Putting something else in mstartupaction like KEYBOARD "{ALT+F}" works OK. Thanks to Dejan Jocic (100355,1742) for this one.

Table definition

```
***** Table setup for USRLVL *****
CREATE TABLE 'USRLVL' (CID C(1) NOT NULL DEFAULT newid(), ;
                       CDESCRIPTION C(35) NOT NULL, ;
                       MSTARTUPACTION M NOT NULL)
***** Create each index for USRLVL *****
INDEX ON CDESCRIPTION TAG DESCRIPTIO
ALTER TABLE 'USRLVL' ADD PRIMARY KEY CID TAG PRIMARY
```

Table Employee

No trick here, just plain stuff : a cid, a usrLvlId as foreign key, a cNom that holds the employee's name, a cTitre (French for cTitle) that holds the employee's position in the company, and a cPassword. The cPassword is not encrypted as of now.

Table definition

```
***** Table setup for EMPLOYEE *****
CREATE TABLE 'EMPLOYEE' (CID       C(6)  NOT NULL DEFAULT newid(), ;
                         CUSRLVLID C(1)  NOT NULL                , ;
                         CNOM      C(30) NOT NULL                , ;
                         CTITRE    C(30) NOT NULL                , ;
                         CPASSWORD C(8)  NOT NULL)

***** Create each index for EMPLOYEE *****
ALTER TABLE 'EMPLOYEE' ADD PRIMARY KEY CID TAG PRIMARY
INDEX ON CUSRLVLID TAG USRLVL
INDEX ON CNOM TAG NOM
```

Set the 1/M Relation

```
************** Begin Relations Setup **************
ALTER TABLE 'EMPLOYEE' ADD FOREIGN KEY TAG USRLVL REFERENCES USRLVL TAG PRIMARY
```

New Views

These views will be used by the new bizness objects that will be created later in this document. They are fairly easy to implement and code. They will be added to the application Database Container.

For those who plan to use their security module as a standard feature, they might as well add this functionality to their QSTART.APP. If you add your views directly to the GENERIC project you'll end up with an error in the new project because the view definition won't reflect the Database change.

- *open the QSTART project*
- *modify QSTARTMA.PRG, Proc MAKEPROJ as follows*

```
*-- Rename DBC and update link to ID field
  WAIT CLEAR
  =MESSAGEBOX([About to rename database. The back-link information for ] + ;
    [the ID, Employee and UserLvl tables must be updated to reflect the new ] + ;
    [database name. Please answer "Yes" to the dialog that ] + ;
    [immediately follows this one.], ;
    MB_ICONINFORMATION, APPNAME_LOC)
  CD DATA
  RENAME generic.dbc TO (lcProject + '.DBC')
  RENAME generic.dcx TO (lcProject + '.DCX')
  RENAME generic.dct TO (lcProject + '.DCT')
  OPEN DATA (lcProject)
  USE ID            && Back link dialog will be displayed at this point

*-- CHANGE - JCM - January 07, 1997 - 14:43:33
*-- update back link of security tables
  USE EMPLOYEE
  USE USRLVL

  PRIVATE lcProjectEmployee
  lcProjectEmployee = lcProject+ "!Employee"
*-- add the view definitions here
***************** View setup for LV_PASSWORD ***************

  CREATE SQL VIEW "LV_PASSWORD" ;
    AS SELECT    EMPLOYEE.CID,  EMPLOYEE.CNOM,  EMPLOYEE.CPASSWORD  FROM
&lcProjectEmployee  WHERE  EMPLOYEE.CID = ?vp_cId

  DBSetProp('LV_PASSWORD', 'View', 'UpdateType', 1)
  DBSetProp('LV_PASSWORD', 'View', 'WhereType', 3)
  DBSetProp('LV_PASSWORD', 'View', 'FetchMemo', .T.)
  DBSetProp('LV_PASSWORD', 'View', 'SendUpdates', .T.)
  DBSetProp('LV_PASSWORD', 'View', 'UseMemoSize', 255)
  DBSetProp('LV_PASSWORD', 'View', 'FetchSize', 100)
  DBSetProp('LV_PASSWORD', 'View', 'MaxRecords', -1)
  DBSetProp('LV_PASSWORD', 'View', 'Tables',lcProjectEmployee )
  DBSetProp('LV_PASSWORD', 'View', 'Prepared', .F.)
  DBSetProp('LV_PASSWORD', 'View', 'CompareMemo', .T.)
  DBSetProp('LV_PASSWORD', 'View', 'FetchAsNeeded', .F.)
  DBSetProp('LV_PASSWORD', 'View', 'FetchSize', 100)
  DBSetProp('LV_PASSWORD', 'View', 'ParameterList', "vp_cId,'C'")
  DBSetProp('LV_PASSWORD', 'View', 'Comment', "Used in ChangePasswordObj")
  DBSetProp('LV_PASSWORD', 'View', 'BatchUpdateCount', 1)
  DBSetProp('LV_PASSWORD', 'View', 'ShareConnection', .F.)

*!* Field Level Properties for LV_PASSWORD
```

```
* Props for the LV_PASSWORD.cid field.
  DBSetProp('LV_PASSWORD.cid', 'Field', 'KeyField', .T.)
  DBSetProp('LV_PASSWORD.cid', 'Field', 'Updatable', .T.)
  DBSetProp('LV_PASSWORD.cid', 'Field', 'UpdateName', lcProjectEmployee +".CID")
  DBSetProp('LV_PASSWORD.cid', 'Field', 'Caption', "CID")
  DBSetProp('LV_PASSWORD.cid', 'Field', 'Comment', "Unique record identifyer")
  DBSetProp('LV_PASSWORD.cid', 'Field', 'DataType', "C(6)")
  DBSetProp('LV_PASSWORD.cid', 'Field', 'DefaultValue', "newid()")
* Props for the LV_PASSWORD.cnom field.
  DBSetProp('LV_PASSWORD.cnom', 'Field', 'KeyField', .F.)
  DBSetProp('LV_PASSWORD.cnom', 'Field', 'Updatable', .F.)
  DBSetProp('LV_PASSWORD.cnom', 'Field', 'UpdateName', lcProjectEmployee + ".CNOM")
  DBSetProp('LV_PASSWORD.cnom', 'Field', 'Caption', "CNOM")
  DBSetProp('LV_PASSWORD.cnom', 'Field', 'Comment', "User name")
  DBSetProp('LV_PASSWORD.cnom', 'Field', 'DataType', "C(30)")
* Props for the LV_PASSWORD.cpassword field.
  DBSetProp('LV_PASSWORD.cpassword', 'Field', 'KeyField', .F.)
  DBSetProp('LV_PASSWORD.cpassword', 'Field', 'Updatable', .T.)
  DBSetProp('LV_PASSWORD.cpassword', 'Field', 'UpdateName', lcProjectEmployee
+".CPASSWORD")
  DBSetProp('LV_PASSWORD.cpassword', 'Field', 'Caption', "CPASSWORD")
  DBSetProp('LV_PASSWORD.cpassword', 'Field', 'Comment', "User Password")
  DBSetProp('LV_PASSWORD.cpassword', 'Field', 'DataType', "C(8)")
***************** View setup for LV_EMPLOYEE ***************

  CREATE SQL VIEW "LV_EMPLOYEE" ;
    AS SELECT    EMPLOYEE.*  FROM   &lcProjectEmployee ORDER BY  EMPLOYEE.CNOM

  DBSetProp('LV_EMPLOYEE', 'View', 'UpdateType', 1)
  DBSetProp('LV_EMPLOYEE', 'View', 'WhereType', 3)
  DBSetProp('LV_EMPLOYEE', 'View', 'FetchMemo', .T.)
  DBSetProp('LV_EMPLOYEE', 'View', 'SendUpdates', .T.)
  DBSetProp('LV_EMPLOYEE', 'View', 'UseMemoSize', 255)
  DBSetProp('LV_EMPLOYEE', 'View', 'FetchSize', 100)
  DBSetProp('LV_EMPLOYEE', 'View', 'MaxRecords', -1)
  DBSetProp('LV_EMPLOYEE', 'View', 'Tables', lcProjectEmployee)
  DBSetProp('LV_EMPLOYEE', 'View', 'Prepared', .F.)
  DBSetProp('LV_EMPLOYEE', 'View', 'CompareMemo', .T.)
  DBSetProp('LV_EMPLOYEE', 'View', 'FetchAsNeeded', .F.)
  DBSetProp('LV_EMPLOYEE', 'View', 'FetchSize', 100)
  DBSetProp('LV_EMPLOYEE', 'View', 'Comment', "Used by our EmployeeObj")
  DBSetProp('LV_EMPLOYEE', 'View', 'BatchUpdateCount', 1)
  DBSetProp('LV_EMPLOYEE', 'View', 'ShareConnection', .F.)

*!* Field Level Properties for LV_EMPLOYEE
* Props for the LV_EMPLOYEE.cid field.
  DBSetProp('LV_EMPLOYEE.cid', 'Field', 'KeyField', .T.)
  DBSetProp('LV_EMPLOYEE.cid', 'Field', 'Updatable', .T.)
  DBSetProp('LV_EMPLOYEE.cid', 'Field', 'UpdateName', lcProjectEmployee +".CID")
  DBSetProp('LV_EMPLOYEE.cid', 'Field', 'Caption', "CID")
  DBSetProp('LV_EMPLOYEE.cid', 'Field', 'Comment', "Unique record identifyer")
  DBSetProp('LV_EMPLOYEE.cid', 'Field', 'DataType', "C(6)")
  DBSetProp('LV_EMPLOYEE.cid', 'Field', 'DefaultValue', "newid()")
```

```
* Props for the LV_EMPLOYEE.cusrlvlid field.
  DBSetProp('LV_EMPLOYEE.cusrlvlid', 'Field', 'KeyField', .F.)
  DBSetProp('LV_EMPLOYEE.cusrlvlid', 'Field', 'Updatable', .T.)
  DBSetProp('LV_EMPLOYEE.cusrlvlid', 'Field', 'UpdateName', lcProjectEmployee
+".cusrlvlid")
  DBSetProp('LV_EMPLOYEE.cusrlvlid', 'Field', 'DataType', "C(1)")
* Props for the LV_EMPLOYEE.cnom field.
  DBSetProp('LV_EMPLOYEE.cnom', 'Field', 'KeyField', .F.)
  DBSetProp('LV_EMPLOYEE.cnom', 'Field', 'Updatable', .T.)
  DBSetProp('LV_EMPLOYEE.cnom', 'Field', 'UpdateName', lcProjectEmployee +".CNOM")
  DBSetProp('LV_EMPLOYEE.cnom', 'Field', 'Caption', "CNOM")
  DBSetProp('LV_EMPLOYEE.cnom', 'Field', 'Comment', "User name")
  DBSetProp('LV_EMPLOYEE.cnom', 'Field', 'DataType', "C(30)")
* Props for the LV_EMPLOYEE.ctitre field.
  DBSetProp('LV_EMPLOYEE.ctitre', 'Field', 'KeyField', .F.)
  DBSetProp('LV_EMPLOYEE.ctitre', 'Field', 'Updatable', .T.)
  DBSetProp('LV_EMPLOYEE.ctitre', 'Field', 'UpdateName', lcProjectEmployee +".CTITRE")
  DBSetProp('LV_EMPLOYEE.ctitre', 'Field', 'Caption', "CTITRE")
  DBSetProp('LV_EMPLOYEE.ctitre', 'Field', 'Comment', "User Title")
  DBSetProp('LV_EMPLOYEE.ctitre', 'Field', 'DataType', "C(30)")
* Props for the LV_EMPLOYEE.cpassword field.
  DBSetProp('LV_EMPLOYEE.cpassword', 'Field', 'KeyField', .F.)
  DBSetProp('LV_EMPLOYEE.cpassword', 'Field', 'Updatable', .T.)
  DBSetProp('LV_EMPLOYEE.cpassword', 'Field', 'UpdateName', lcProjectEmployee
+".CPASSWORD")
  DBSetProp('LV_EMPLOYEE.cpassword', 'Field', 'Caption', "CPASSWORD")
  DBSetProp('LV_EMPLOYEE.cpassword', 'Field', 'Comment', "User Password" )
  DBSetProp('LV_EMPLOYEE.cpassword', 'Field', 'DataType', "C(8)")

*-- ENDCHANGE - JCM - January 07, 1997 - 15:34:11

  CLOSE DATA
  CD ..
```

· *rebuild the App*

View : lv_Password

This view will be used in our ChangePasswordBizObj.

View Definition

See above code where you'll replace lcProjectEmployee by <yourProjectName> + "!Employee"

View : lv_Employee

This view will be used by our EmployeeObj.

View Definition

See above code where you'll replace lcProjectEmployee by <YourApplicationClassName> + "!Employee"

Add 2 records to Id.DBF

Add 2 entries to Id.DBF, so that the CID fields of our new tables get incremented properly.

Type	Keyname	Value	IncrementProcedure	MaxLength
C	USRLVL	1	IncrementBase62(LEFT(id.value, id.maxlength))	1
C	EMPLOYEE	1	IncrementBase62(LEFT(id.value, id.maxlength))	6

Modifications to aDataEnv.prg

These are fairly easy to implement.

Once again, you might want to change your aDataEnv.prg in GENERIC.PJX to have your new applications reflect this functionality.

```
DEFINE CLASS LocalTablesEnvironment AS <appclass>Environment
  FUNCTION LoadCursors()
    DIMENSION this.aCursors[2]
    this.aCursors[1] = "UsrLvl"
       this.aCursors[2] = "Employee"
       ENDFUNC
ENDDEFINE

DEFINE CLASS PasswordEnvironment AS <appclass>Environment
  FUNCTION loadcursors()
    DIMENSION this.aCursors[1]
    this.aCursors[1] = "v_Password"
  ENDFUNC
ENDDEFINE

DEFINE CLASS EmployeeEnvironment AS <appclass>Environment
  FUNCTION loadCursors()
    DIMENSION this.aCursors[1]
    this.aCursors[1] = "v_Employee"
  ENDFUNC
ENDDEFINE




DEFINE CLASS LoginEnvironment AS <appclass>Environment
  FUNCTION LOADCURSORS
    DIMENSION this.aCursors[2]
    this.aCursors[1] = "Employee"
    this.aCursors[2] = "UsrLvl"
  ENDFUNC
ENDDEFINE

DEFINE CLASS UsrLvl AS cTableCursor
  CursorSource = "UsrLvl"
ENDDEFINE

DEFINE CLASS Employee AS cTableCursor
  CursorSource = "Employee"
ENDDEFINE

DEFINE CLASS v_Password AS cDynamicViewCursor
  cCursorSource = "v_Password"
  NoDataOnLoad = .F. && default behavior = .T.
  *-- this avoids to requery() the first time
ENDDEFINE

DEFINE CLASS v_Employee AS cDynamicViewCursor
  cCursorSource = "v_Employee"
```

```
  NoDataOnLoad = .F. && default behavior = .T.
  *-- this avoids to requery() the first time
ENDDEFINE
```

If you want to use this code for an existing project, don't forget to replace <appclass> by your application class name.

And now, to the fun part !

CLASSLIB : CLOGIN.VCX

Introduction

CLOGIN is a class library that contains the necessary classes to add a basic security system to your CodeBook 3.0 applications. It may therefore be kept safely in your CDBK30\COMMON30\LIBS. We've decided to pack all our security related classes into a single library. You might well decide to proceed in a different manner, adding the menu bars to CTMENUS.VCX, etc.

Figure 2 : the cLogin Classlib structure as displayed by the VFP5.0 class browser

Class : AdminEmployeeBar

AdminEmployeeBar is a cBar that gives the system administrator access to the employee table. It might be created with the menu builder.

The original code has been written in French. From here on, I'll add the '--EN :' comment with a probable <g> English translation.*

```
**************************************************
*-- Class:        adminemployeebar (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  cbar (c:\cdbk30\common30\libs\cmenus.vcx)
*-- BaseClass:    container
*-- Barre de menu "Utilisateurs du système"
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS adminemployeebar AS cbar
       Width = 118
       Height = 25
       ccaption = "\<Utilisateurs du système"
       *--EN : "\<System Users"
       cmessage = "Définit les utilisateurs du système"
       *--EN : "Define System Users"
       cskipfor = "! EMPTY(WONTOP())"
       Name = "adminemployeebar"

       PROCEDURE Click
              = DoForm ("EmployeeForm")
       ENDPROC
ENDDEFINE
```

```
*
*-- EndDefine: adminemployeebar
************************************************
```

Class : AdminLoginBar

AdminLoginBar is a cBar that gives users access to the login form. Watch the Click() method: if the user level has changed, all the menu bars that might have been hidden are showed, then eventually removed according to the user level

```
************************************************
*-- Class:        adminloginbar (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  cbar (c:\cdbk30\common30\libs\cmenus.vcx)
*-- BaseClass:    container
*-- Barre de menu "Ouvrir une nouvelle session"
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS adminloginbar AS cbar


       Width = 118
       Height = 25
       ccaption = "\<Ouvrir une nouvelle session"
       *--EN : "\<Login"
   cskipfor = "!EMPTY(WONTOP())"
       Name = "adminloginbar"


       PROCEDURE Click
              LOCAL lcUserLevel

              lcUserLevel = goApp.GetUserLevel()
              = goApp.Login()

              IF goApp.GetUserLevel() # lcUserLevel
                goApp.ShowBars()
                goApp.HideBars()
              ENDIF
       ENDPROC


ENDDEFINE
*
*-- EndDefine: adminloginbar
************************************************
```

Class : AdminPasswordBar

It is a cBar that gives the user access to the change password form.

```
**********************************************
*-- Class:        adminpasswordbar (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  cbar (c:\cdbk30\common30\libs\cmenus.vcx)
*-- BaseClass:    container
*-- Barre de menu "changer le mot de passe"
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS adminpasswordbar AS cbar


       Width = 118
       Height = 25
       ccaption = "\<Changer le mot de passe"
       *--EN : "\<Change password"
       cmessage = "Change le mot de passe"
       *--EN : "Change the password"
       cskipfor = "!EMPTY(WONTOP())"
       Name = "adminpasswordbar"

       PROCEDURE Click
              = DoForm("ChangePasswordForm")
       ENDPROC
ENDDEFINE
*
*-- EndDefine: adminpasswordbar
**********************************************
```

Get the menu working

What you need to do is create subclasses of the 3 bars that we defined, and store them into the aMenus class library. This can be done easily with the Menu Builder. Click on the aMenus classlib in the Project Manager, double click on the mainmenu, right click and select builder. Position the mouse cursor on the ‘Administration ‘ menu pad, right click and select ‘Add Bar... ‘ This gives you access to the Menu Bar Item Builder. Click on the New Class button.

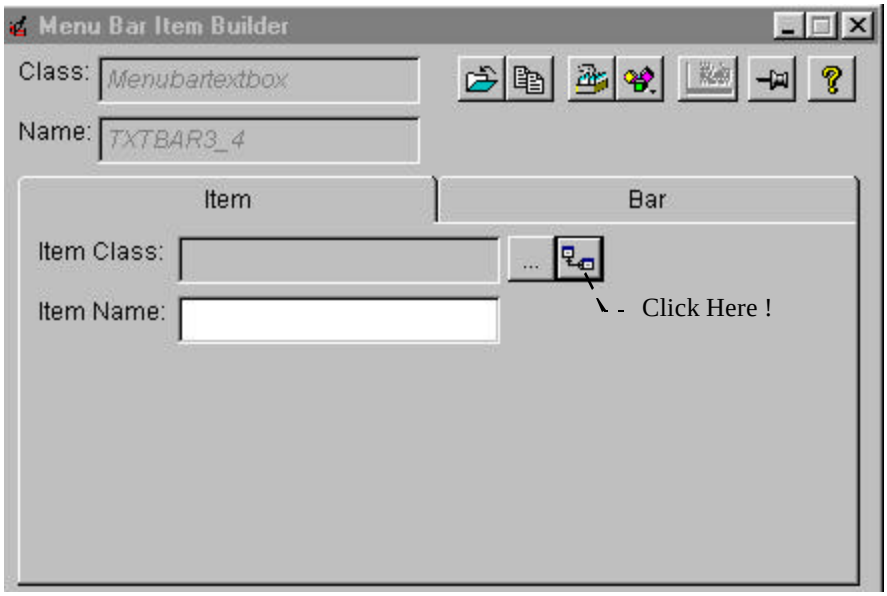


Figure 3 : The Menu builder Item Bar Builder

The new class dialog will appear. Be extremely careful, especially in the third dialog. You need to store your new classes in the application aMenus Classlib, while the Menu Builder insists in storing them in the generic ctMenus Classlib.

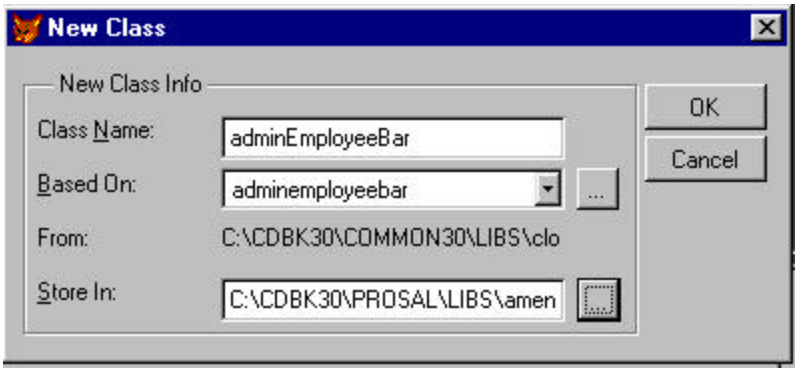


Figure 4 : Create a subclass of the AdminEmployeeBar in aMenus

GENERIC.PJX: create a subclass of the AdminLoginBar, AdminPassword and AdminEmployeeBar in the Generic aMenus class library.

Code

If you decide not to use the menu builder, here is the code...

```
*************************************************
*-- Class:        adminemployeebar (c:\cdbk30\generic\libs\amenus.vcx)
*-- ParentClass:  adminemployeebar (c:\cdbk30\common30\libs\clogin.vcx)
*-- BaseClass:    container
```

```
*
DEFINE CLASS adminemployeebar AS adminemployeebar
 Name = "adminemployeebar"
ENDDEFINE
*
*-- EndDefine: adminemployeebar
**********************************************

**********************************************
*-- Class:        adminloginbar (c:\cdbk30\generic\libs\amenus.vcx)
*-- ParentClass:  adminloginbar (c:\cdbk30\common30\libs\clogin.vcx)
*-- BaseClass:    container
*
DEFINE CLASS adminloginbar AS adminloginbar
  Name = "adminloginbar"
ENDDEFINE
*
*-- EndDefine: adminloginbar
**********************************************
```

```
*************************************************
*-- Class:        adminpasswordbar (c:\cdbk30\generic\libs\amenus.vcx)
*-- ParentClass:  adminpasswordbar (c:\cdbk30\common30\libs\clogin.vcx)
*-- BaseClass:    container
*
DEFINE CLASS adminpasswordbar AS adminpasswordbar
  Name = "adminpasswordbar"
ENDDEFINE
*
*-- EndDefine: adminpasswordbar
*************************************************

*************************************************
*-- Class:        administrationpopup (c:\cdbk30\generic\libs\amenus.vcx)
*-- ParentClass:  cadministrationpopup (c:\cdbk30\common30\libs\ctmenus.vcx)
*-- BaseClass:    container
*-- The  menu class for the Administration menu popup.
*
#INCLUDE "c:\cdbk30\autosoft\include\appincl.h"
*
DEFINE CLASS administrationpopup AS cadministrationpopup
  Width = 173
  Height = 25
  Name = "administrationpopup"

PROCEDURE loadchildren
  *-- The following code was generated by Menu Builder:
  *:ClassLibrary amenus.vcx

  #DEFINE NUMCHILDREN 4
  LOCAL laChildren[NUMCHILDREN,ALEN(this.aChildren,2)]
  laChildren[1,CHILD_CLASS] = "Adminloginbar"
  laChildren[1,CHILD_NAME]  = "oAdminloginbar"
  laChildren[2,CHILD_CLASS] = "Adminpasswordbar"
  laChildren[2,CHILD_NAME]  = "oAdminpasswordbar"
  laChildren[3,CHILD_CLASS] = "Adminemployeebar"
  laChildren[3,CHILD_NAME]  = "oAdminemployeebar"
  laChildren[4,CHILD_CLASS] = "Adminseparatorbar"
  laChildren[4,CHILD_NAME]  = "oAdminseparatorbar"

  DIMENSION this.aChildren[NUMCHILDREN,ALEN(laChildren,2)]
  =ACOPY(laChildren,this.aChildren)

  RETURN NUMCHILDREN
ENDPROC

ENDDEFINE
*
*-- EndDefine: administrationpopup
```

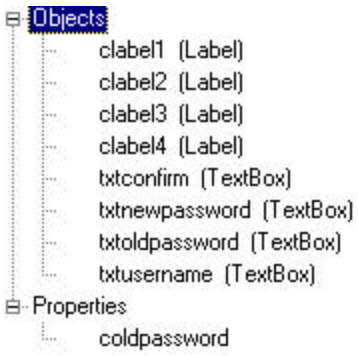
Hide and show items on the toolbar

The only thing you'll have to do is create a custom init() method:

```
DODEFAULT()

LOCAL lcUserLevel
lcUserLevel = GoApp.GetUserLevel()

*-- depending on the user level, turn off access to the buttons
*-- in this example, cmdxx is the button you want to hide

IF lcUserLevel # USER_APPDEV_LOC and lcUserLevel # USER_OPSMGR_LOC
  this.cmdxx.Visible = .F.
ENDIF
```

Class : ChangePasswordObj

ChangePasswordObj is a BizObj responsible for retrieving the password pertaining to the current user and storing a new one. cDELoader.cDataEnvironment is set to PasswordEnvironment. (see definition in aDataEnv.prg)



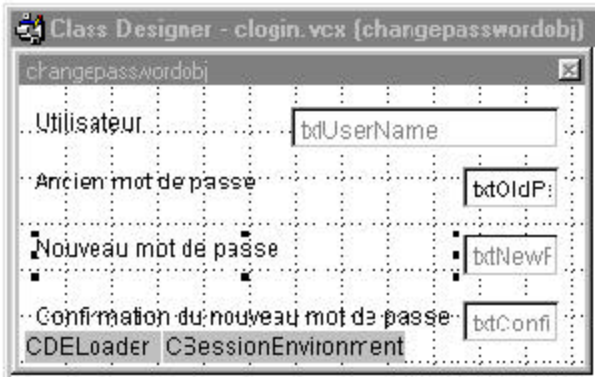


Figure 5 : ChangePasswordObj

```
**************************************************
*-- Class:        changepasswordobj (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  cbizobj (c:\cdbk30\common30\libs\cbizness.vcx)
*-- BaseClass:    container
*-- Our change password business object
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS changepasswordobj AS cbizobj


  Width = 295
  Height = 147
  Name = "changepasswordobj"
  oSessionEnvironment.Clabel1.Height = 16
  oSessionEnvironment.Clabel1.Left = 1
```

```
oSessionEnvironment.Clabel1.Top = 0
oSessionEnvironment.Clabel1.Width = 131
oSessionEnvironment.Clabel1.Name = "Clabel1"
oSessionEnvironment.Top = 45
oSessionEnvironment.Left = 12
oSessionEnvironment.Width = 126
oSessionEnvironment.Height = 16
oSessionEnvironment.TabIndex = 0
oSessionEnvironment.Name = "oSessionEnvironment"
ODELOADER.ODELOADER.Height = 22
ODELOADER.ODELOADER.Left = 0
ODELOADER.ODELOADER.Top = 0
ODELOADER.ODELOADER.Width = 80
ODELOADER.ODELOADER.Name = "ODELOADER"
ODELOADER.Top = 60
ODELOADER.Left = 12
ODELOADER.Width = 71
ODELOADER.Height = 16
ODELOADER.TabIndex = 0
ODELOADER.cdataenvironment = "PasswordEnvironment"
ODELOADER.Name = "ODELOADER"

*-- the user's old password
coldpassword = .F.

ADD OBJECT txtoldpassword AS ctextbox WITH ;
     Height = 22, ;
     Left = 232, ;
     TabIndex = 2, ;
     Top = 43, ;
     Width = 50, ;
     PasswordChar = "*", ;
     Name = "txtOldPassWord"

ADD OBJECT txtnewpassword AS ctextbox WITH ;
     Enabled = .F., ;
     Height = 22, ;
     Left = 232, ;
     TabIndex = 3, ;
     Top = 78, ;
     Width = 50, ;
     PasswordChar = "*", ;
     Name = "txtNewPassword"

ADD OBJECT clabel1 AS clabel WITH ;
     Caption = "Confirmation du nouveau mot de passe", ;
     Height = 22, ;
     Left = 8, ;
     Top = 113, ;
     Width = 220, ;
     TabIndex = 0, ;
     Name = "Clabel1"
     *--EN : "Confirm new password"
```

```
ADD OBJECT clabel2 AS clabel WITH ;
     Caption = "Ancien mot de passe", ;
     Left = 8, ;
     Top = 43, ;
     Width = 220, ;
     TabIndex = 0, ;
     Name = "Clabel2"
     *--EN : "Old password"
```

```
ADD OBJECT clabel3 AS clabel WITH ;
     Caption = "Nouveau mot de passe", ;
     Left = 8, ;
     Top = 78, ;
     Width = 220, ;
     TabIndex = 0, ;
     Name = "Clabel3"
     *--EN : "New password"

ADD OBJECT txtconfirm AS ctextbox WITH ;
     Enabled = .F., ;
     Height = 22, ;
     Left = 232, ;
     TabIndex = 4, ;
     Top = 113, ;
     Width = 50, ;
     PasswordChar = "*", ;
     Name = "txtConfirm"

ADD OBJECT clabel4 AS clabel WITH ;
     Caption = "Utilisateur", ;
     Left = 8, ;
     Top = 12, ;
     TabIndex = 0, ;
     Name = "Clabel4"
     *--EN : "User"

ADD OBJECT txtusername AS ctextbox WITH ;
     ControlSource = "v_Password.cNom", ;
     Enabled = .F., ;
     Left = 142, ;
     TabIndex = 1, ;
     Top = 12, ;
     Width = 140, ;
     Name = "txtUserName"

PROCEDURE ODELOADER.Init
     vp_cId= goApp.GetEmployeeId()
     Cdeloader::Init()
ENDPROC

PROCEDURE ODELOADER.Destroy
     this.Parent.cOldpassword = v_password.cPassword
     Cdeloader::Destroy()
ENDPROC


PROCEDURE txtoldpassword.InteractiveChange
     LOCAL llEnabled

     llEnabled = (UPPER(ALLTRIM(this.Parent.cOldPassWord)) ==
                  UPPER(ALLTRIM(this.Value)))
     this.Parent.txtNewPassword.Enabled = llEnabled
```

```
      this.Parent.txtConfirm.Enabled = llEnabled
  ENDPROC


ENDDEFINE
*
*-- EndDefine: changepasswordobj
**********************************************
```

Class : BaseChangePasswordForm

BaseChangePasswordForm is a cBizObjRetValForm. It has the ChangePasswordObj dropped onto it. There is a cLabel that says "Saisissez votre nouveau mot de passe et confirmez-le" (**-- EN : "Input your new password and confirm").

Be sure to check the Validate() method of the form: if no password is entered, ask user if they want to cancel. If yes, TABLEREVERT() otherwise set focus to the old password. If new password is empty, warn user that it may not be so and set focus to the new password. If the confirmation password differs from the new password, warn user...

The Ok command button Click() method does a TABLEUPDATE() then releases the form, while the Cancel command button Click() method does a TABLEREVERT() then releases the form. Don't forget that the ???_LOC have been defined in Strinsg.H and Strings.DBF

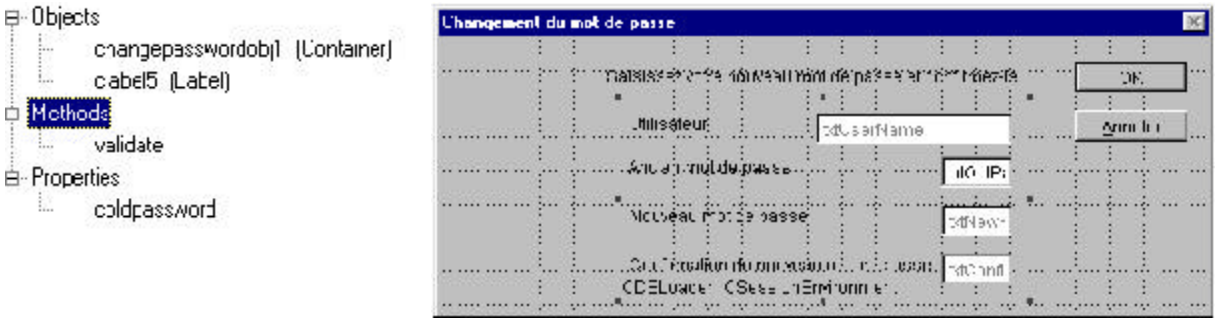


Figure 6 : BaseChangePasswordForm

```
**************************************************
*-- Class:        basechangepasswordform (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  cbizobjretvalform (c:\cdbk30\common30\libs\cforms.vcx)
*-- BaseClass:    form
*-- Our finished "change password" class
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS basechangepasswordform AS cbizobjretvalform

  Height = 198
  Width = 551
  DoCreate = .T.
  Caption = "Changement du mot de passe"
  Name = "basechangepasswordform"
  refBizObj.Top = 0
  refBizObj.Left = 0
  refBizObj.Width = 38
  refBizObj.Height = 18
  refBizObj.Name = "refBizObj"
  cmdOK.Top = 18
  cmdOK.Left = 456
  cmdOK.TabIndex = 2
```

```
cmdOK.Name = "cmdOK"
cmdCancel.Top = 53
cmdCancel.Left = 456
cmdCancel.TabIndex = 3
cmdCancel.Name = "cmdCancel"

*-- the user's old password
coldpassword = .F.

ADD OBJECT clabel5 AS clabel WITH ;
     Caption = "Saisissez votre nouveau mot de passe et confirmez-le", ;
     Height = 22, ;
     Left = 120, ;
     Top = 21, ;
     Width = 323, ;
     TabIndex = 0, ;
     Name = "Clabel5"

ADD OBJECT changepasswordobj1 AS changepasswordobj WITH ;
     Top = 29, ;
     Left = 128, ;
     Width = 295, ;
     Height = 164, ;
     Name = "Changepasswordobj1", ;
     Clabel2.Name = "Clabel2", ;
     oSessionEnvironment.Clabel1.Name = "Clabel1", ;
     oSessionEnvironment.Top = 129, ;
     oSessionEnvironment.Left = 75, ;
     oSessionEnvironment.Width = 126, ;
     oSessionEnvironment.Height = 16, ;
     oSessionEnvironment.Name = "oSessionEnvironment", ;
     refBizObj.Top = 0, ;
     refBizObj.Left = 216, ;
     refBizObj.Width = 51, ;
     refBizObj.Height = 15, ;
     refBizObj.Name = "refBizObj", ;
     txtOldPassWord.Name = "txtOldPassWord", ;
     txtNewPassword.Name = "txtNewPassword", ;
     Clabel1.Name = "Clabel1", ;
     Clabel3.Name = "Clabel3", ;
     txtConfirm.Name = "txtConfirm", ;
     oDELoader.oDELoader.Name = "oDELoader", ;
     oDELoader.Top = 129, ;
     oDELoader.Left = 3, ;
     oDELoader.Width = 71, ;
     oDELoader.Height = 16, ;
     oDELoader.Name = "oDELoader", ;
     txtUserName.Name = "txtUserName", ;
     Clabel4.Name = "Clabel4"


*-- Validate the password change
PROCEDURE validate
```

```
IF !thisform.oBizObj.txtNewPassword.Enabled
  *{ Commented at 15:32:01 on November 06, 96
   * IF YesNo(NOPSWDENTERED_LOC) = IDNO && error has to be .T. or .F.
    IF !YesNo(NOPSWDENTERED_LOC)
   *} End Commenting 15:32:01 - November 06, 96
    =TABLEREVERT()
    thisform.release()
  ELSE
    thisform.oBizObj.txtOldPassWord.Value = ""
    thisform.oBizObj.txtOldPassWord.SetFocus()
  ENDIF
  RETURN .F.
ENDIF
```

```
      IF EMPTY(thisform.oBizObj.txtNewPassword.value)
        = ErrorMsg(PASSWORDEMPTY_LOC)
        thisform.oBizObj.txtNewPassword.Setfocus()
        RETURN .F.
      ENDIF

      IF thisform.oBizObj.txtConfirm.Value # thisform.oBizObj.txtNewPassword.value
        = ErrorMSg(PSWDNOTCNFRM_LOC)
        thisform.ObizObj.txtconfirm.value = ""
        thisform.oBizObj.txtConfirm.SetFocus()
        RETURN .F.
      ENDIF
  ENDPROC


  PROCEDURE cmdOK.Click
      IF thisform.validate()
        *-- CHANGE - JCM - November 06, 1996 - 15:25:18
        *-- grossière erreur...
        * REPLACE employee.cpassword WITH thisform.ObizObj.txtconfirm.value
        REPLACE v_Password.cPassword WITH thisform.ObizObj.txtconfirm.value
        *-- ENDCHANGE
        = TABLEUPDATE()
        thisform.release()
      ENDIF
  ENDPROC


  PROCEDURE cmdCancel.Click
      = TABLEREVERT()
      thisform.Release()
  ENDPROC


ENDDEFINE
*
*-- EndDefine: basechangepasswordform
**************************************************
```

Class: cLogin

cLogin is a modalRetvalForm providing basic Login facilities. It allows the user to select his name from a combobox and then enter his password.

Figure 7: cLogin

```
************************************************
*-- Class:        clogin (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  cmodalretvalform (c:\cdbk30\common30\libs\cforms.vcx)
*-- BaseClass:    form
*-- Base login container.  Allows entry of name and password
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS clogin AS cmodalretvalform

  Height = 118
  Width = 311
  DoCreate = .T.
  Caption = "Ouverture d'une nouvelle session"
  *-- EN: "Login"
  *-- name of the field that holds the password in the employee table
  cpassword = "cPassWord"
  *-- Name or the table that holds user information, such as name and password
  ctable = "Employee"
  *-- Tag name used to search the employee table for the user name
  ctagname = "Primary"
  *-- Name of field that holds the user name, in the form "cname, cId"
  cfieldname = "cNom, cId"
  Name = "clogin"
  cmdOK.Top = 87
  cmdOK.Left = 58
  cmdOK.TabIndex = 3
  cmdOK.Name = "cmdOK"
  cmdCancel.Top = 87
  cmdCancel.Left = 153
  cmdCancel.TabIndex = 4
  cmdCancel.Name = "cmdCancel"


  ADD OBJECT clabel1 AS clabel WITH ;
```

```
     Caption = "Utilisateur", ;
     Left = 24, ;
     Top = 12, ;
     TabIndex = 0, ;
     Name = "Clabel1"
     *-- EN: "User"

ADD OBJECT clabel2 AS clabel WITH ;
     Caption = "Mot de passe", ;
     Left = 24, ;
     Top = 48, ;
     TabIndex = 0, ;
     Name = "Clabel2"
     *-- EN: "Password"

ADD OBJECT txtpassword AS ctextbox WITH ;
     Height = 22, ;
     Left = 120, ;
     TabIndex = 2, ;
     Top = 44, ;
     Width = 50, ;
     PasswordChar = "*", ;
     Name = "txtPassWord"

ADD OBJECT cboname AS ccombobox WITH ;
     BoundColumn = 2, ;
     RowSourceType = 3, ;
     Height = 22, ;
     Left = 120, ;
     TabIndex = 1, ;
     Top = 12, ;
     Width = 175, ;
     Name = "cboName"

PROCEDURE Load
     IF ! USED(this.cTable)
       USE this.cTable
     ENDIF
     SELECT (this.cTable)
ENDPROC

PROCEDURE Unload
     Cmodalretvalform::Unload()

     IF USED("cNames")
       USE IN cNames
     ENDIF

     IF USED(this.cTable)
       USE IN this.cTable
     ENDIF
ENDPROC
```

```
  PROCEDURE Refresh
       **-- setup our Work areas

       LOCAL lnOldSelect, ;
             lcFldName, ;
             lcUserId, ;
             luRetVal

       lnOldSelect = SELECT()
       SELECT (thisform.cTable)
       lcFldName = "cId"
       lcUserId = thisform.cboName.Value

       *-- move the record pointer to the appropriate record
       luRetVal = LOOKUP(&lcFldName, lcUserId, ;
                &lcFldName, thisform.cTagName)

       *-- In this case, if lookup fails, it returns an empty string
       *-- because we look up a character field

       SELECT (lnOldSelect)
       IF ! EMPTY(luRetval)
              RETURN .T.
       ELSE
              RETURN .F.
       ENDIF
  ENDPROC

  PROCEDURE cmdOK.Click
       *-- now, check the password

       IF UPPER(ALLTRIM(EVAL(this.parent.cPassword))) ==
UPPER(ALLTRIM(this.parent.txtPassword.Value))
          thisform.hide()
       ELSE
              =ErrorMsg(BADPASSWORD_LOC)
              this.parent.txtPassword.Value=""
              this.parent.txtPassword.SetFocus()
       ENDIF
  ENDPROC

  PROCEDURE cmdCancel.Click
       thisform.uRetVal = .F.
       thisform.hide()
  ENDPROC

  PROCEDURE cboname.Init
       this.RowSource = "SELECT " + thisForm.cFieldName + ;
                        " FROM " + thisForm.cTable + ;
                        " ORDER BY " + thisForm.cFieldName + ;
                        " INTO CURSOR cNames"
       this.ListIndex = 1
       Ccombobox::Init()
```

```
  ENDPROC


  PROCEDURE cboname.Destroy
       Ccombobox::Destroy()
       IF USED("cNames")
              USE IN cNames
       ENDIF
  ENDPROC


  PROCEDURE cboname.InteractiveChange
       thisform.refresh()
  ENDPROC

ENDDEFINE
*
*-- EndDefine: clogin
**************************************************
```

Class: BaseLoginForm

BaseLoginForm is a specialized cLogin. We have added two fields, some labels and one method, GetUserLevel() that returns the user level as set in the UsrLvl table.

If the user has entered the right password it will be possible to click the Ok button. When the Ok button Click() fires, it will return the Employee Id (cid) and its associated level to the calling program, the goApp.Login(), that will then process the returned string and store the Employee Id into GoApp.CEmployeeId and its user level in GoApp.cUserLevel. Based on the cUserLevel property of the application object, we'll be able to decide which menu bars he cannot see and what he'll be able to do with our application administration forms. Based on the cEmployeeId property, we could devise some sort of log file.

If the user enters a wrong password, he has no other choice but to cancel. The click() method of the cancel command button will return an empty string to goApp.Login() that will then decide to abort the application.

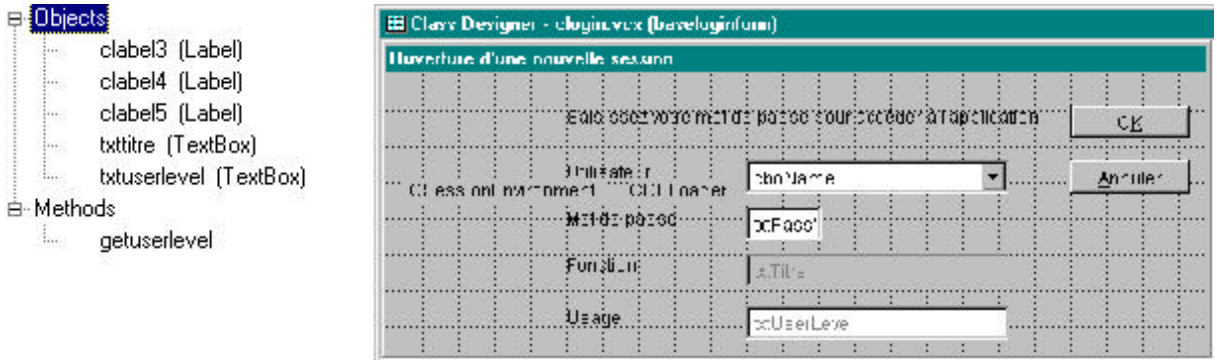


Figure 8 : BaseLoginForm

You might find that the txtUserLevel text box breaks somewhat our security scheme since any user can see how many "Usage level" there are. Feel free to remove this object if you please.

```
**************************************************
*-- Class:        baseloginform (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  clogin (c:\cdbk30\common30\libs\clogin.vcx)
*-- BaseClass:    form
*-- Our finished login screen
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS baseloginform AS clogin

  Height = 186
  Width = 547
  DoCreate = .T.
  Name = "baseloginform"
  oSessionEnvironment.Clabel1.Name = "Clabel1"
  oSessionEnvironment.Top = 70
  oSessionEnvironment.Left = 12
  oSessionEnvironment.Width = 136
  oSessionEnvironment.Height = 16
```

```
oSessionEnvironment.Name = "oSessionEnvironment"
oDELoader.oDELoader.Name = "oDELoader"
oDELoader.Top = 70
oDELoader.Left = 162
oDELoader.Width = 81
oDELoader.Height = 16
oDELoader.Name = "oDELoader"
cmdOK.Top = 21
cmdOK.Left = 456
cmdOK.TabIndex = 3
cmdOK.Name = "cmdOK"
cmdCancel.Top = 58
cmdCancel.Left = 456
cmdCancel.TabIndex = 4
cmdCancel.Name = "cmdCancel"
Clabel1.Left = 120
Clabel1.Top = 58
Clabel1.TabIndex = 0
Clabel1.Name = "Clabel1"
Clabel2.Left = 120
Clabel2.Top = 89
Clabel2.TabIndex = 0
Clabel2.Name = "Clabel2"
txtPassWord.Left = 240
txtPassWord.TabIndex = 2
txtPassWord.Top = 89
txtPassWord.Name = "txtPassWord"
cboName.Left = 240
cboName.TabIndex = 1
cboName.Top = 58
cboName.Name = "cboName"
ADD OBJECT clabel3 AS clabel WITH ;
     Caption = "Fonction", ;
     Height = 22, ;
     Left = 120, ;
     Top = 120, ;
     Width = 61, ;
     TabIndex = 0, ;
     Name = "Clabel3"

ADD OBJECT clabel4 AS clabel WITH ;
     Caption = "Usage", ;
     Height = 22, ;
     Left = 120, ;
     Top = 155, ;
     Width = 79, ;
     TabIndex = 0, ;
     Name = "Clabel4"

ADD OBJECT txttitre AS ctextbox WITH ;
     Enabled = .F., ;
     Left = 240, ;
     TabIndex = 0, ;
```

```
     Top = 120, ;
     Width = 175, ;
     Name = "txtTitre"

ADD OBJECT txtuserlevel AS ctextbox WITH ;
     Enabled = .F., ;
     Left = 240, ;
     TabIndex = 0, ;
     Top = 155, ;
     Width = 175, ;
     DisabledBackColor = RGB(255,255,255), ;
     Name = "txtUserLevel"
ADD OBJECT clabel5 AS clabel WITH ;
     Caption = "Saisissez votre mot de passe pour accéder à l'application", ;
     Height = 22, ;
     Left = 120, ;
     Top = 21, ;
     Width = 323, ;
     TabIndex = 0, ;
     Name = "Clabel5"

*-- Returns the user level as set in the UsrLvl table
PROCEDURE getuserlevel
     LOCAL llCloseUsrLvl , ;
           lcUserLevel

               IF ! USED("UsrLvl")
                 *-- CHANGE - JCM - February 03, 1997 - 21:30:59
                 *-- specify the path
                 *-- USE UsrLvl IN 0
                 USE GoApp.cDefaultDataBase + "!UsrLvl" IN 0
                 *-- ENDCHANGE - JCM - February 03, 1997 - 21:31:47
                 llCloseUsrLvl = .T.
               ENDIF

     lcUserLevel = LOOKUP(UsrLvl.cDescription, ;
                          Employee.cUsrLvlId, ;
                          UsrLvl.cId, ;
                          "PRIMARY")
     IF llCloseUsrLvl
       USE IN UsrLvl
            ENDIF
     RETURN lcUSerLevel
ENDPROC

PROCEDURE Init
     IF Clogin::Init()
       thisform.Refresh()
     ENDIF
ENDPROC

PROCEDURE Refresh
     IF Clogin::Refresh()
```

```
        thisform.txtTitre.Value = cTitre
        thisform.txtUserLevel.Value = thisform.GetUserLevel()
      ELSE
        STORE "" TO ;
        thisform.txtTitre.value, ;
        thisform.txtUserLevel.Value
      ENDIF
ENDPROC

PROCEDURE cmdOK.Click
      LOCAL llCloseUsrLvl
      clogin.cmdOk::Click()
      thisform.uRetVal = cid + "," + thisform.GetUserLevel()
ENDPROC
```

```
  PROCEDURE cmdCancel.Click
       cLogin.cmdCancel::click()
       thisform.uRetval = ""
  ENDPROC

ENDDEFINE
*
*-- EndDefine: baseloginform
**************************************************
```

Class: EmployeeObj

EmployeeObj is a standard BizObj that holds the employee information. cDELoader has its cDataEnvironment set to "EmployeeEnvironment" (see aDataEnv.PRG)

Figure 9: EmployeeObj

```
**************************************************
*-- Class:        employeeobj (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  cbizobj (c:\cdbk30\common30\libs\cbizness.vcx)
*-- BaseClass:    container
*-- Holds the employee information (password and user level)
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS employeeobj AS cbizobj

  Width = 331
  Height = 156
  Name = "employeeobj"
  oSessionEnvironment.Clabel1.Height = 16
  oSessionEnvironment.Clabel1.Left = 1
  oSessionEnvironment.Clabel1.Top = 0
  oSessionEnvironment.Clabel1.Width = 131
  oSessionEnvironment.Clabel1.Name = "Clabel1"
  oSessionEnvironment.Top = 96
  oSessionEnvironment.Left = 0
```

```
oSessionEnvironment.Width = 126
oSessionEnvironment.Height = 16
oSessionEnvironment.Name = "oSessionEnvironment"
oDELoader.oDELoader.Height = 22
oDELoader.oDELoader.Left = 0
oDELoader.oDELoader.Top = 0
oDELoader.oDELoader.Width = 80
oDELoader.oDELoader.Name = "oDELoader"
oDELoader.Top = 120
oDELoader.Left = 0
oDELoader.Width = 71
oDELoader.Height = 16
oDELoader.cdataenvironment = "EmployeeEnvironment"
oDELoader.Name = "oDELoader"

ADD OBJECT txtoldpassword AS ctextbox WITH ;
     ControlSource = "v_Employee.cPassword", ;
     Height = 22, ;
     Left = 132, ;
     TabIndex = 2, ;
     Top = 51, ;
     Width = 50, ;
     PasswordChar = "", ;
     Name = "txtOldPassWord"

ADD OBJECT clabel2 AS clabel WITH ;
     Caption = "Mot de passe", ;
     Height = 22, ;
     Left = 16, ;
     Top = 51, ;
     Width = 93, ;
     TabIndex = 0, ;
     Name = "Clabel2"
     *-- EN : "Password"

ADD OBJECT clabel4 AS clabel WITH ;
     Caption = "Utilisateur", ;
     Left = 16, ;
     Top = 20, ;
     TabIndex = 0, ;
     Name = "Clabel4"
     *-- EN : "User"

ADD OBJECT txtusername AS ctextbox WITH ;
     ControlSource = "v_Employee.cNom", ;
     Left = 132, ;
     TabIndex = 1, ;
     Top = 20, ;
     Width = 175, ;
     Name = "txtUserName"

ADD OBJECT clabel3 AS clabel WITH ;
     Caption = "Fonction", ;
```

```
      Height = 22, ;
      Left = 16, ;
      Top = 84, ;
      Width = 61, ;
      TabIndex = 0, ;
      Name = "Clabel3"
      *-- EN : "Title"

  ADD OBJECT txttitre AS ctextbox WITH ;
      ControlSource = "v_Employee.cTitre", ;
      Left = 132, ;
      TabIndex = 0, ;
      Top = 81, ;
      Width = 175, ;
      Name = "txtTitre"

  ADD OBJECT clabel1 AS clabel WITH ;
      Caption = "Niveau", ;
      Left = 16, ;
      Top = 112, ;
      Name = "Clabel1"
      *-- EN : "Level"

  ADD OBJECT cbouserlevel AS ccombobox WITH ;
      BoundColumn = 2, ;
      RowSourceType = 3, ;
      ControlSource = "v_Employee.cUsrLvlId", ;
      FirstElement = 1, ;
      Left = 132, ;
      Top = 112, ;
      Width = 175, ;
      Name = "cboUserLevel"

  PROCEDURE cbouserlevel.Init
      Ccombobox::Init()
      this.RowSource = "SELECT cDescription, cid FROM UsrLvl ORDER BY cDescription INTO
CURSOR cUsrLvl"
  ENDPROC

  PROCEDURE cbouserlevel.Destroy
      Ccombobox::Destroy()
      IF USED("cUsrLvl")
            USE IN cUsrLvl
      ENDIF
  ENDPROC


ENDDEFINE
*
*-- EndDefine: employeeobj
**************************************************
```

Class: EmployeeForm

EmployeeForm is a cBizObjNoCritForm. This is a class that we added to the CodeBook cForms class library that will be discussed further in this document. It is nothing more than a mainteance form with only two tabs. The selection tab has been removed. It is plain standard design. You might as well want to use the CodeBook cBizObjMaintForm. All you'd have to do is change the lv_Employee definition to add search criterias.

The EmployeeObj has been dropped on the data entry tab and the pgfNoCritBizObj.Page2.grdList.grcList.ControlSource set to "v_Employee.cNom"

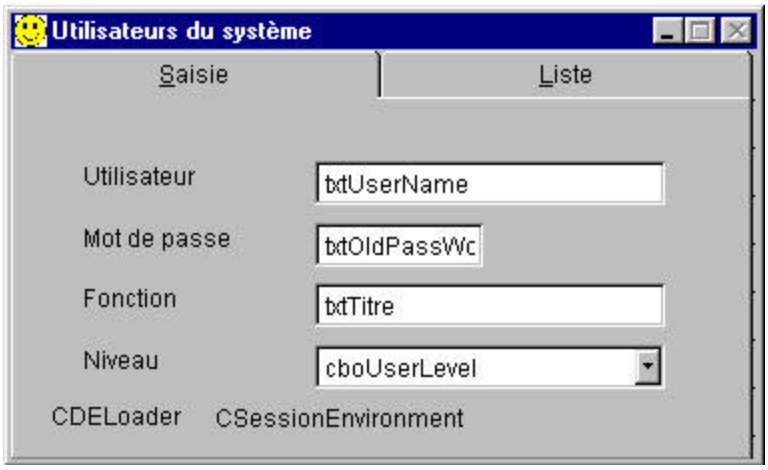


Figure 10: EmployeeForm

Class: SecuritySticker

SecuritySticker is a cLabel that you can drop on any cBizObjForm. This one has been designed to make a cBizObjForm readonly for system users and guests by setting the form lAllowDelete, lAllowNew and lAllowsave properties to .F. when such users are logged in.

Be creative : design more security stickers suited to your needs !

```
**************************************************
*-- Class:        securitysticker (c:\cdbk30\common30\libs\clogin.vcx)
*-- ParentClass:  clabel (c:\cdbk30\common30\libs\ccontrls.vcx)
*-- BaseClass:    label
*-- Our base security sticker. Responsible for setting the security options
*-- of a form.
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS securitysticker AS clabel


  BackColor = RGB(255,0,0)  && Flame Red!
  BackStyle = 1
  Caption = "Sécurité"
  * -- EN : "Security"
```

```
Height = 16
Visible = .F.
Width = 51
Name = "securitysticker"
```

```
  PROCEDURE Init
       *------------------------- Copyright ------------------------
       *  Author............: José Constant
       *  Project...........: CIEF
       *  Created...........: 06/08/96  13:57:11
       *  Copyright.........: (c) Constant Software Systems, 1996
       *---------------------- Usage Section ----------------------
       *) Description.......: restrict user access based to their userlevel
       *)                   : application developper and application
       *)                   : administrator are given all the rights
       *)                   : system users (USER_COMMON_LOC) and invited persons
       *)                   : (USER_INVITE_LOC) have a limited access to certain
       *)                   : areas.  Simply drop an instance of this sticker,
       *)                   : or a subclass of it onto the form you want to
       *)                   : protect.

       LOCAL lcUserLevel
       lcUserLevel = goApp.GetUserLevel()

       IF lcUserLevel = USER_COMMON_LOC OR lcUserLevel = USER_INVITE_LOC
         thisform.lAllowdelete = .F.
         thisform.lAllowNew = .F.
         thisform.lAllowSave= .F.
       ENDIF
       this.release()
  ENDPROC
ENDDEFINE
*
*-- EndDefine: securitysticker
**************************************************
```

ClassLib : aLogin.vcx

This library is specific to the application. The classes offer no special enhancements compared to their parents. They are there to allow you to bring in some customization, like a bitmap.

You might want to define these classes in your GENERIC.PJX. You'd simply have to drop a bitmap on them once in the new project...

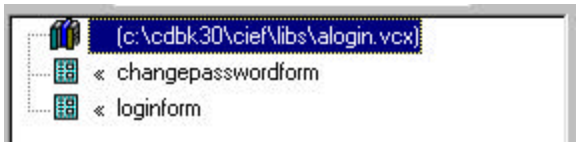


Figure 11 : the aLogin Classlib structure as shown by the VFP5.0 class browser

Class : ChangePasswordForm

ChangePasswordForm is a BaseChangePasswordForm with a bitmap on it.

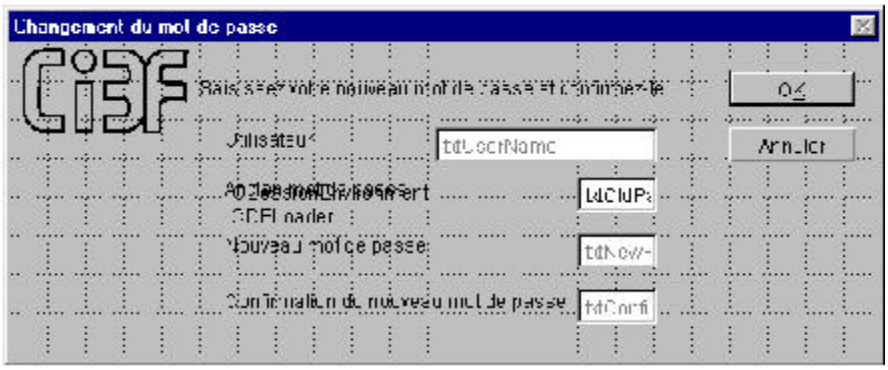


Figure 12: ChangePasswordForm

Class : LoginForm

LoginForm is a BaseLoginForm with a bitmap on it

Figure 13 : LoginForm

ClassLib : cForms

We have added two specialized classes to the CodeBook cForms library, cModalRetValForm and cBizObjNoCritForm

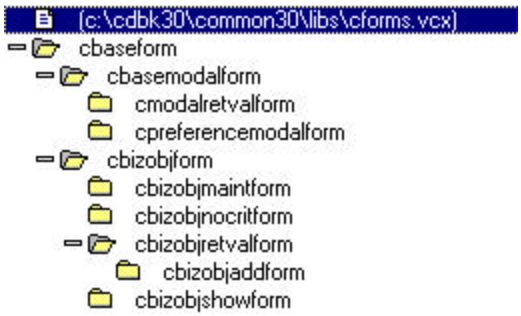


Figure 14 : the modified cForms ClassLib as displayed by the VFP 3.0 class browser

Class : cModalRetValForm

cModalRetValForm is a specialized cBaseModalForm that returns a value through its GetRetVal() method.

```
************************************************
*-- Class:        cmodalretvalform (c:\cdbk30\common30\libs\cforms.vcx)
*-- ParentClass:  cbasemodalform (c:\cdbk30\common30\libs\cforms.vcx)
*-- BaseClass:    form
*-- A form with no bizobj on it that returns a value.
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS cmodalretvalform AS cbasemodalform


  DoCreate = .T.
  Caption = "cModalRetValForm"
  Name = "cmodalretvalform"

  ADD OBJECT cmdok AS ccommandbutton WITH ;
       Top = 216, ;
       Left = 108, ;
       Caption = "O\<K", ;
       Default = .T., ;
       Name = "cmdOK"

  ADD OBJECT cmdcancel AS ccommandbutton WITH ;
       Top = 216, ;
       Left = 203, ;
       Cancel = .T., ;
       Caption = "\<Annuler", ;
       Name = "cmdCancel"
       *-- EN : "\<Cancel"
```

```
  *-- called by the cmdOk::click() method to load the uRetVal property of
  *-- the form
  PROCEDURE getretval
  ENDPROC

  PROCEDURE cmdok.Click
       thisform.GetRetVal()
       thisform.Hide()
  ENDPROC

  PROCEDURE cmdcancel.Click
       thisform.uRetVal = ""
       thisform.Hide()
  ENDPROC

ENDDEFINE
*
*-- EndDefine: cmodalretvalform
**************************************************
```

Class cBizObjNoCritForm

This is a cBizObjForm, but contrary to the cBizObjMainForm, it has only two tabs. The selection tab has been removed. This design makes sense when you have to select one record from a limited set of records.

```
**************************************************
*-- Class:        cbizobjnocritform (c:\cdbk30\common30\libs\cforms.vcx)
*-- ParentClass:  cbizobjform (c:\cdbk30\common30\libs\cforms.vcx)
*-- BaseClass:    form
*-- Maintenance form with no criteria.  Requires NoDataOnLoad=.F. in View Cursor
*
#INCLUDE "c:\cdbk30\common30\include\framincl.h"
*
DEFINE CLASS cbizobjnocritform AS cbizobjform


  Height = 240
  Width = 373
  DoCreate = .T.
  Caption = "CBizObjNoCritForm"
  MaxButton = .F.
  KeyPreview = .T.
  lreleaseonesc = .T.
  Name = "cbizobjnocritform"

  ADD OBJECT pgfnocritbizobj AS cpageframe WITH ;
       ErasePage = .T., ;
       Top = 0, ;
       Left = 0, ;
       Width = 372, ;
       Height = 240, ;
       Name = "pgfNoCritBizObj", ;
       Page1.Caption = "\<Saisie", ;
```

```
Page1.BackColor =
   (EVAL("RGB("+SUBSTR(RGBSCHEME(15,1),AT(",",RGBSCHEME(15,1),3)+1))), ;
Page1.Name = "Page1", ;
Page2.Caption = "\<Liste", ;
Page2.BackColor =
   (EVAL("RGB("+SUBSTR(RGBSCHEME(15,1),AT(",",RGBSCHEME(15,1),3)+1))), ;
Page2.Name = "Page2"
*-- EN : "\<Data Entry"
*-- EN : "\<List"
```

```
ADD OBJECT cbizobjnocritform.pgfnocritbizobj.page2.grdlist AS cgrid WITH ;
     ColumnCount = 1, ;
     DeleteMark = .F., ;
     GridLines = 1, ;
     HeaderHeight = 0, ;
     Height = 168, ;
     Left = 28, ;
     ReadOnly = .T., ;
     ScrollBars = 2, ;
     Top = 23, ;
     Width = 312, ;
     Name = "grdList", ;
     Column1.FontBold = .F., ;
     Column1.Width = 10, ;
     Column1.ReadOnly = .T., ;
     Column1.Name = "grcList"


ADD OBJECT cbizobjnocritform.pgfnocritbizobj.page2.grdlist.grclist.grhlist ;
          AS header WITH ;
     FontBold = .F., ;
     Caption = "Header1", ;
     Name = "grhList"


ADD OBJECT cbizobjnocritform.pgfnocritbizobj.page2.grdlist.grclist.txtlist ;
           AS textbox WITH ;
     FontBold = .F., ;
     BackColor = RGB(255,255,255), ;
     BorderStyle = 0, ;
     ForeColor = RGB(0,0,0), ;
     Margin = 0, ;
     ReadOnly = .T., ;
     ColorSource = 3, ;
     Name = "txtList"

PROCEDURE copyright
     *---------------------- Location Section ----------------------
     *  Library...........: Common30\cForms
     *  Class.............: Cbizobjnocritform
     *  Method............: CopyRight()
     *------------------------- Copyright ------------------------
     *  Author............: José Constant
     *  Project...........: CIEF
     *  Created...........: 01/05/96  19:23:58
     *  Copyright.........: (c) Constant Software Systems         , 1996
     *----------------------- Usage Section ------------------------
     *$ Usage.............: Drop your bizobj on the "Saisie" frame
     *)                   : Be sure to define the control source of the grid yourself!
     *% Example...........:

ENDPROC
```

```
PROCEDURE new
     *-- Autoselect the data entry page
     LOCAL lnDataEntryPage
     IF Cbizobjform::New()= FILE_OK
            lnDataEntryPage = this.pgfnoCritBizObj.GetPageNumber(PAGE_DATAENTRY)
            IF this.pgfNoCritBizObj.ActivePage # lnDataEntryPage
                   this.pgfNoCritBizObj.ActivePage = lnDataEntryPage
            ENDIF
     ENDIF
ENDPROC
```

```
  PROCEDURE KeyPress
       LPARAMETERS nKeyCode, nShiftAltCtrl
       LOCAL lnSelectionPage
       *-- Allow the enter key to act as a selection key for the list grid
       IF nKeyCode = KEY_ENTER AND this.ActiveControl.Name = "grdList"
         this.pgfNoCritBizObj.ActivePage = ;
            this.pgfNoCritBizObj.GetPageNumber(PAGE_DATAENTRY)
         NODEFAULT
         *-- CHANGE - JCM - September 09, 1996 - 15:18:07
                *-- added a requery to get the data entry tab to refresh
                thisform.requery()
       ELSE
         CBizObjForm::KeyPress(nKeyCode, nShiftAltCtrl)
       ENDIF
  ENDPROC


  PROCEDURE pgfnocritbizobj.Init
       *-- Make the page frame fit nicely into the surrounding form
       this.Height = this.Parent.Height - 1
       this.Width = this.Parent.Width - 1
  ENDPROC


  PROCEDURE grdlist.Init
       Cgrid::Init()
       this.grcList.width= this.Width-10
  ENDPROC


  PROCEDURE txtlist.RightClick
       *-- right mouse click sends user back to data entry (same as Enter Key)
       thisform.pgfNoCritBizObj.ActivePage = ;
            thisform.pgfNoCritBizObj.GetPageNumber(PAGE_DATAENTRY)
       *-- CHANGE - JCM - July 29, 1996 - 13:32:09
       *-- added a refresh to get the title to refresh
       thisform.refresh()
       NODEFAULT
  ENDPROC


ENDDEFINE
*
*-- EndDefine: cbizobjnocritform
**************************************************
```

ClassLib : Aapp.VCX

Class : Application object

You might want to touch the Generic class in Aapp.VCX of your Generic.PJX to have your new applications reflect this functionality.

The base CodeBook application class has been enhanced to bring in the security capability. We have added two protected properties, cUserLevel and cEmployeeId. Based on the cUserLevel property of the application object, we'll be able to decide which menu bars the users cannot see and what they'll be able to do with our application administration forms. Based on the cEmployeeId property, we could devise some sort of log file.

We have added seven new methods and a new property

- GetUserLevel() returns the current user level
- GetEmployeeId() returns the user ID
- Login() logs the user in
- GetStartupAction() defines the action to be taken at startup
- ShowBars() add menu bars that might have been hidden previously; also makes toolbar buttons visible if they had been hidden
- HideBars() hides menu bars and toolbar buttons according to the user level;
- Custom protected method GetDefaultDatabase(): it will return the fullpath to the database, by example C:\DATA\CIEF. Otherwise, the security extension as described above would have problems working from data located elsewhere than in a \data subdirectory. We need to provide path to the DBC for setup and security things
- Custom property, cDefaultDatabase, holds the value returned by GetDefaultDatabase()

We added code to the BeforeReadEvents() method to execute the action to be taken at startup

We modified:

- the ShowMenu() method to accomodate a call to ShowBars() and HideBars()
- the Do() method

```
**********************************************
*-- Class:        Cief (c:\cdbk30\Cief\libs\aapp.vcx)
*-- ParentClass:  capplication (c:\cdbk30\common30\libs\capp.vcx)
*-- BaseClass:    container
*-- The application class.
*
#INCLUDE "c:\cdbk30\Cief\include\appincl.h"
*
DEFINE CLASS Cief AS capplication

*-- some code removed for clarity

  Width = 139
  Height = 34
  csplashimage = "SplashImage"
  Name = "Cief"
```

```
*-- the user level of the currently logged in user
PROTECTED cuserlevel

*-- the Id of the currently logged in user
PROTECTED cemployeeid

*-- returns the current user level
PROCEDURE getuserlevel
     RETURN this.cUserLevel
ENDPROC

*-- Returns the EmployeeId
PROCEDURE getemployeeid
     RETURN this.cEmployeeID
ENDPROC

*-- Logs the user in
PROCEDURE login
     LOCAL lcEmployeeID, ;
                         lcUserLevel, ;
                         lcLoginString

     *-- Save the current values of these vars in case user is logging in
     *-- again but decides to cancel
     lcEmployeeID = this.cEmployeeID
     lcUserLevel = this.cUserLevel
     lcLoginString = DoForm ("loginForm")

     this.cEmployeeID = LEFT(lcLoginString, AT(",", lcLoginString) - 1)
     this.cUserLevel = SUBSTR(lcLoginString, AT(",", lcLoginString) + 1)

     IF EMPTY(this.cUserLevel)
       this.cEmployeeID = lcEmployeeID
       this.cUserLevel = lcUserLevel
     ENDIF

     *-- The user level is what determines if the user successfully logged
     *-- in or not, and determines what menu pads are shown.
     RETURN !EMPTY(this.cUserLevel)
ENDPROC

*-- Defines the action to be taken at startup
PROCEDURE getstartupaction
     *-- Returns the action to take based on the user level
     *-- The action is just a Visual FoxPro command stored as a character
     *-- string in the mStartupAction field of the usrlvl table.

     LOCAL lnOldArea, ;
           lcAction, ;
           llCloseWhenDone
```

```
     lnOldArea = SELECT()

     IF !USED("usrlvl")
       *-- CHANGE - JCM - February 03, 1997 - 21:30:59
       *-- specify the path
       *-- USE UsrLvl IN 0
       USE GoApp.cDefaultDataBase + "!UsrLvl" IN 0
       *-- ENDCHANGE - JCM - February 03, 1997 - 21:31:47
       llCloseWhenDone = .T.
     ENDIF

     SELECT usrlvl
     lcAction = LOOKUP(mStartupAction, ;
                    this.cUserLevel, ;
                    cdescription, ;
                    "DESCRIPTIO")

     IF llCloseWhenDone
       USE IN usrlvl
     ENDIF

     SELECT (lnOldArea)

     RETURN lcAction
ENDPROC

PROCEDURE showbars
     *-- Show menu bars that might have been hidden when the user was
     *-- logged in with a lower User Level

     *-- the preferences form
     this.oMenu.oFilePad.oPopUp.oFileUserPrefBar.Show()
     *-- etc ...

     *-- the toolbar buttons
     *-- in the example, cmdxx is the toolbar button to show
     this.oMainToolbar.cmdxx.visible = .F.
ENDPROC

PROCEDURE hidebars
     *-- Hides menu bars when the user logs in with a low User Level

     LOCAL lcUserLevel

     lcUserLevel= goApp.GetUserLevel()

     IF lcUserLevel # USER_APPDEV_LOC and lcUserLevel # USER_OPSMGR_LOC
       *-- the preferencesform
       this.oMenu.oFilePad.oPopUp.oFileUserPrefBar.Hide()
       *-- etc ...

       *-- the toolbar buttons
```

```
       *-- in the example, cmdxx is the toolbar button to hide
       IF TYPE("this.oMainToolbar") = "O"
          This.oMainToolbar.cmdxx.visible = .F.
       ENDIF
     ENDIF
ENDPROC


PROCEDURE beforereadevents
     LOCAL lcAction

     lcAction = this.getStartupAction()

     IF ! EMPTY(lcAction)
       &lcAction
     ENDIF
ENDPROC

 PROCEDURE showmenu
     Capplication::Showmenu()
     this.HideBars()
ENDPROC

 PROCEDURE GetDefaultDatabase
    *-- CHANGE - JCM - November 04, 1996 - 20:12:54
    *-- adapted from cDataEnv.GetDefaultDatabase

    LOCAL lcDefaultDatabase, ;
          lcKey, ;
          lUseLocalData

    lcDefaultDatabase = ""

    IF UPPER(this.GetSetting("Data Settings","Use Local Data","YES"))=="YES"
      lcKey = DBCLOCATION_LOCALKEY
      lUseLocalData = .T.
    ELSE
      lcKey = DBCLOCATION_REMOTEKEY
      lUseLocalData = .F.
    ENDIF

    lcDefaultDatabase = ALLT(this.GetSetting(lcKey, APPCLASS))

    IF RIGHT(lcDefaultDatabase, 1) <> "\"
      lcDefaultDatabase= lcDefaultDatabase+ "\"
    ENDIF

    lcDefaultDatabase = lcDefaultDatabase + APPCLASS

    RETURN FULLPATH(lcDefaultDatabase)
 ENDPROC

 PROCEDURE DO
```

```
*-- Override to ensure that user preference settings have been
*-- entered. We cannot put this code in the Init() since PreferenceForm
*-- needs to access methods of the application object, which would
*-- not yet exist.
*{ Commented at 10:17:05 on November 07, 96
* IF EMPTY(this.GetSetting("Data Settings", "Use Local Data", ""))
*   *-- Need to specify settings
*   ?? CHR(7)
*   =MESSAGEBOX(SELECTPREFERENCE_LOC, MB_ICONINFORMATION, APPNAME_LOC)
*   IF DoForm("preferenceform")
*     RETURN CApplication::Do()
*   ELSE
*     this.Release()
*   ENDIF
* ELSE
*   RETURN CApplication::Do()
* ENDIF

IF EMPTY(this.GetSetting("Data Settings", "Use Local Data", ""))
  *-- Need to specify settings
  ?? CHR(7)
  =MESSAGEBOX(SELECTPREFERENCE_LOC, MB_ICONINFORMATION, APPNAME_LOC)
  IF ! DoForm("preferenceform")
    this.Release()
  ENDIF
ENDIF
*-- CHANGE - JCM - November 07, 1996 - 10:17:26
*-- removed from the init() method and placed here
*-- since in the init() the application object is not
*-- yet defined
*-- CHANGE - JCM - November 04, 1996 - 21:49:42
this.cDefaultDataBase = this.GetDefaultDatabase()
*-- ENDCHANGE

IF ! FILE(DEBUGMODEFILE)
  llRetVal = this.login()
ELSE
  this.cEmployeeID = ""
  this.cUserLevel = USER_APPDEV_LOC
  llRetVal = .T.
ENDIF

IF ! llRetVal
  this.Cleanup()
ELSE
  *-- CHANGE - JCM - January 16, 1997 - 20:03:59
  *-- tell user with which data they work
  LOCAL lcDataPath
  lcDataPath = SUBSTR(this.cDefaultDataBase, 1, ;
                      rat("\", this.cDefaultDataBase) -1)
  _SCREEN.Caption = this.GetCaption() + " [" + lcDataPath + "]"
  *-- ENDCHANGE - JCM - January 16, 1997 - 20:09:04
  RETURN CApplication::Do()
```

```
   ENDIF

   *} End Commenting 10:17:05 - November 07, 96
  ENDPROC
ENDDEFINE
*

*-- EndDefine: prosal
**********************************************
```

Included class library

The code reflects all the changes described earlier. The business objects are in the line of the eBizObj class by Ed Leafe, except that the eBizObj changes were made directly to my cBizObj class. Following the same design decision, I've enhanced my cBizObjForm with a RefBizObj.

What you should do is probably make a copy of a standard Codebook form 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.

If you decide to take that road, please bear in mind that you'll have to slightly adapt the following code.

Conclusion

Well, that's all there is to it ! Pretty straightforward I think. Don't forget to send me your comments, they'll be appreciated and answered in a timely fashion.

 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, fax (+32)(0)87/67.41.89, www.club.innet.be/~2396.