THE CODEBOOK NEWS

July 1997
Volume 1, Issue 1

TABLE OF CONTENTS

Introduction

Yair Alan Griver

Has it only been two years?

It seems like a lifetime ago that Paul Bienick, Menachem Bazian, David Blumenthal, Robert Godbey and I began work on what was to become "The Visual FoxPro 3.0 Codebook."

We wanted to give a synopsis of what our thinking was in regards to application development and where it fits into the operation of a business. It was a “here’s where we are, and this is what it looks like today” kind of effort. We had (and still have) a firm belief that application development only succeeds when it is focused on the business, and that compartmentalizing application development into three logical layers would allow us maximum flexibility in our efforts.

Two years later, with the growth of COM and the quickly maturing DCOM, organizing an application into component sections that offer well defined services to each other is becoming a standard approach in development circles.

I am often asked if I would change things if I was rewriting Codebook today. Simply put, the answer is "yes." I would have used more object passing as a way of standardizing parameter passing between layers. I also would have written a better Section Three of the book, where I discuss the framework itself.

One of the challenges of writing a software book is that the publisher wants a final copy within weeks (if not days) of the product’s ship date. One of the truisms of training is that the classes typically only fill up weeks after the product ships. As a result, I learned how to explain the classes and approaches in the Codebook framework only after the book was available in stores.

...and that’s where the wonderful thing called "The Codebook Community" comes in.

Through efforts like this newsletter, Visual FoxExpress and Codebook For Mere Mortals (both the guide and the framework), Codebook knowledge is being made available by the fine efforts of talented individuals throughout the world. Codebook was a labor of love - and it’s nice to see that love being compounded by all of your efforts.

I hope that you enjoy reading this newsletter every month -- I’m sure that I will.

Yair Alan Griver is a principal in Flash Creative Management, a consulting and software development firm in Hackensack, New Jersey. Flash specializes in business process reengineering, custom software solutions, and also offers training in FoxPro and many other software products. He is Contributing Editor of FoxPro Advisor and Data Based Advisor and author of the Visual FoxPro 3.0 Codebook. (201) 489-2500, CompuServe 71541,3150.

The Codebook Newsletter Begins

The first newsletter dedicated to Codebook Professionals World Wide

Charles T. Blankenship

The Concept

The Codebook News has begun! The purpose of this newsletter is to provide a place where Codebook professionals can gather together and trade knowledge critical to Codebook development.

Thanks to Yair Alan Griver and the other Flashers at Flash Creative Management, Inc., an incredibly powerful framework exists that makes the development of highly scalable applications possible. Hats off to Alan. Now, the rest is up to us, the dedicated Codebook lovers.

There is an incredibly bright future for Codebook development, but that future is only what the participants make it. As everyone knows, development that takes place in a group usually produces a finished product that is greater in quality and functionality than one developed in seclusion. In order to take advantage of a group environment, a group must first be created where extensions to the Codebook framework can be proposed, analyzed, designed, developed and tested for public and more importantly free consumption. The Codebook News is hopefully the beginning of that group.

One might argue the point that giving other developers hard earned ideas and technological secrets may erode a company's competitive advantage. After all, there is only so much work in the market for Codebook development. Why equip competitors with hard earned technological secrets, right? While this view is easy to buy into, it is in actuality short sighted. Currently, Codebook development is fragmented. Developer one is modifying Codebook to their advantage and developer two through 400 are doing the same thing. Consider how many user modules have been developed to work with Codebook applications. How many thousands of hours were wasted due to duplicated efforts? How many hours could have been saved if a common, well designed and free Codebook extension had been available? Many!

This Codebook development group, consisting of dedicated Codebook professionals who analyze, design, implement and make available to the general public high quality Codebook extensions, can help to increase Codebook development efficiency dramatically. If Codebook developers band together and establish common extensions, staggering productivity gains can be realized. Sooner or later, the Codebook community will reach a point where applications will be thrown together at an alarming rate with a functionality heretofore unheard of.

This is a critical point. If Codebook developers approach Codebook development as a team, the productivity of the entire team goes up (since our development efforts will be usable and available to all). When this happens, the FoxPro/Codebook development environment will begin to be viewed as a place where incredible productivity gains can be experienced. Once this occurs, the opportunities to provide solutions to corporate America using VFP and Codebook will begin to rise as well. Once this point is reached, more work will be available for all with this platform. We will not loose our "piece of the pie" to another developer, I believe that the "pie" will get much bigger for all. This however, will only happen if productivity is increased to such a level that corporate America can't help but take notice of the high quality, low cost applications that can be developed using this methodology. A dream? Maybe. But what is life without them?

It is commonly known that Microsoft has been lagging in their advertising support of VFP, the Power Builder killer. Together, however, we have the opportunity to build an incredible

development environment that can't help but get the best advertising available, word of mouth. Any division in the Codebook community (or needless duplication of effort) results on a reliance on Microsoft to properly advertise. Let's help them out a little!!! Making this dream come true might wake the giant wake up. Maybe, just maybe, if we do a good enough job, Microsoft may have a reason to stand up and shout to corporate America how wonderful their VFP product is and how they will support it in the future.

Previously VFP developers suffered a blow with the articles that appeared in Infoweak and other national magazines. VFP professionals do not have to sit back and let this bad, unfounded publicity create a self-fulfilling prophecy. However, those who keep their mouths shut and do nothing about this sorry situation, deserve whatever they get.

The Challenge

To accomplish this goal Codebook professionals must dedicate their time and effort to increasing the quality as well as the quantity of Codebook extensions. United we can realize a victory, divided, a mediocre existence continues. Codebook is not a mediocre development tool. Neither is VFP. Both are phenomenal environments that deserve to be promoted as a viable solution to small, medium and large businesses. Together we can make this happen.

There are some steps to consider before this dream can be realized. Codebook has been out for some time now and many modifications have been made to it. Herein lies the problem. In order to develop common enhancements for Codebook, from which all Codebook developers can benefit, standards must be agreed upon that preserve a common development foundation. Once this foundation exists, the enhancements developed by one can be published and guaranteed to operate on any other development firm's version of Codebook.

If a common foundation is not established, Codebook development will remain fragmented, divided and unable to benefit from the synergy and productivity available in a group environment. Therefore, the first task is to define the core classes that will become the foundation of all development. This can be as simple as reaching a common agreement that all controls in the cContrls.VCX are untouchable. That way, if company specific subclasses inherit from a common superclass, the extensions developed in one company can be used by anyone who has an untouched cContrrls.VCX. Second, begin accepting ideas for enhancements and then participate as a group to analyze and design those enhancement ideas and grow them into robust additions to the Codebook Framework.

Codebook itself was developed in a think tank environment at Flash Creative Management where some of the most creative (Hum? Any relation to the way the company name was derived??) minds in the industry participated in its development. The really incredible situation that presents itself is that the creative environment into which Codebook was born can still exist. The only difference being the location of the design meetings are no longer within the hallowed halls of a Hackensack, NJ building but will migrate to the I*Net.

The Medium

The proposal, analysis and design efforts can take place on Ed Leafe's web site. Mr. Leafe has already offered to provide the Web space and services to make this happen. Membership requires only one thing, the desire to develop the most robust VFP development environment on the planet and the determination and dedication required to make it happen.

This Newsletter

The Codebook News itself will accompany the Ed Leafe website effort by providing four functions. First it will be the place where the finished extensions are presented to the Codebook community as a whole. These articles will detail the reason for the new extension, document its functionality, and describe how to integrate it into the Codebook framework so the entire Codebook community can benefit from it.

Second, this newsletter will provide a place to explore the functionality of the existing Codebook classes. Each month, a section of this newsletter will be dedicated to exploring the functionality of these classes. Properties and methods will be explained and examples will detail the *full* functionality of what we already have available in the framework.

Third, this newsletter will provide a place for individuals to illustrate to the Codebook community their own ideas and advancements in the framework and make them available to everyone for the greater good.

Finally, it will have a Tips, Tricks and Traps section exactly like FoxPro Advisor.

Conclusion

It is important to realize that this is a not for profit endeavor. All participation must be pro bono publico, initially. At some point in the future, if all individuals consuming the newsletter agree, a yearly subscription can be instituted that will go to compensate the contributing authors for the time they dedicate to the preparation of their articles. (Each article can easily take 25 hours on average to complete from inception, through coding, into article formulation and completion … it is easy to see why people don't do this for the money).

Currently, Codebook developers do not have a medium to express their creativity. The Codebook News will hopefully provide a means for all the Codebook developers to get their ideas published and into the hands of fellow Codebook lovers where they can be put to good use and *reuse*.

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: cblanke@norfolk.infi.net, www.savvysolutions.com, CompuServe 76132,2575

Enhancement for IsAdding()

The original Codebook rarely resulted in a reference to a table buffered business object being placed into the THISFORM.oBizObj property. Therefore, it is not a surprise that this error has not risen its head until now. However, with the advancements made to Codebook by Ed Leafe with eBizObj, it is now very possible that a table buffered business object can become the primary business object. This enhancement to IsAdding() ensures that this function returns the correct answer to the IsAdding question in any situation, original or enhanced Codebook.

Charles T. Blankenship

Introduction

Recently I noticed a problem with the native IsAdding() Codebook function. The following is a listing of that function as it existed on the original CD-ROM provided with Codebook (reprinted from the original publication with permission).

```
************************************************
*  FUNCTION IsAdding()
************************************************
*  Author...: Paul Bienick
*  Project..: Codebook 3.0
*  Created..: 07/24/95  14:14:20
* Copyright.: (c)Flash Creative Management, Inc., 1995
*) Description.......: Returns .T. if the
*)                     alias specified in
*)                     tcAlias is in the midst of
*)                     adding a new record.
*  Calling Samples...:
*  Parameter List....:
*  Major change list.:
FUNCTION IsAdding(tcAlias)
*-- Returns .T. if the user is in the midst of
*-- adding a record to the alias specified in the
*-- tcAlias parameter.
LOCAL lcGetFldState

IF EMPTY(tcAlias) OR ;
  !EMPTY(tcAlias) AND ;
   CURSORGETPROP("BUFFERING")=DB_BUFOFF
   RETURN .F.
ENDIF

lcGetFldState = GETFLDSTATE(-1, tcAlias)

IF EMPTY(tcAlias) OR;
  !USED(tcAlias) OR ;
   ISNULL(lcGetFldState)
       RETURN .F.
ENDIF
```

```
RETURN ("3" $ lcGetFldState OR "4" $ lcGetFldState)
ENDFUNC
```

The problem occurred when this function was executed against a cursor where table buffering was activated. The condition arose when the user added their record, repositioned their record pointer (off of the newly added record) and then performed an action that triggered the calling of the IsAdding() function. Since the record pointer was no longer positioned on the appended record, none of the characters returned by GETFLDSTATE() were 3's or 4's and IsAdding() erroneously returned a .F. I had a trouble call to fix.

This was a particularly challenging problem. The signature of IsAdding() could not be changed without being forced to modify every call made to IsAdding() throughout the framework ,but this function had to be made aware of table buffering when it was active.

The solution rested with the creation of two new functions and the renaming of the existing one. The original IsAdding( ) code was completely removed and placed in another function with a new name, IsAddingOriginal( ). New code was written to perform the "Is Adding?" test on a table buffered cursor if needed and resides in the IsAddingTB() function … TB for Table Buffering. Finally, a brand new function was written but given the same name as IsAdding(). This is what enabled me to keep the public signature the same while adding table buffering capability.

Listing 1: New implementation for IsAdding()

```
**************************************************
*  FUNCTION IsAddingTB()
**************************************************
* Author............: Charles T. Blankenship
* Project...........: Codebook 5.0
* Created...........: 05/13/97  19:30:00
* Copyright.........: (c) Software Assets of Virginia,
*                         Inc. 1997
*) Description......: Returns .T. if the alias
*)        specified in tcAlias is in the midst of
*)        adding a new record.  This is the same thing
*)        as  the regular IsAdding except for the fact
*)        the fact that it scans through all of the
*)        records checking for new records.  To be
*)        used when  table buffering is in effect.
*  Calling Samples...:
*  Parameter List....:
*  Major change list.:
*-------------------------------------------------

FUNCTION IsAddingTB( tcAlias )
LOCAL lnRecNo, ;
      lcAlias, ;
      llIsAdding
*-------------------------------
*-- Condition the alias parameter
*-------------------------------
IF TYPE( 'tcAlias' ) <> 'C' OR EMPTY( tcAlias )
   lcAlias = ALIAS()
ELSE
   lcAlias = tcAlias
ENDIF
```

```
*-----------------------------------------------------
*-- Store current RP position and initialize return
*-- value
*-----------------------------------------------------
lnRecNo  = RECNO( lcAlias )
llIsAdding = .F.
GO TOP IN ( lcAlias )
*-----------------------------------------------------
*-- Scan through the alias looking for newly added
*-- records
*-----------------------------------------------------
DO WHILE .NOT. llIsAdding
   *=================================================
   llIsAdding = IsAddingOriginal( lcAlias )
   *=================================================
   *-----------------------------------
   *-- Interpret the results of the test
   *-----------------------------------
   IF EOF( lcAlias ) OR llIsAdding
      EXIT
   ENDIF
   *-------------------------------------------
   *-- Process the *next* record, if there is one
   *-------------------------------------------
   IF .NOT. EOF( lcAlias )
      SKIP IN ( lcAlias )
   ENDIF
ENDDO
*-----------------------------------------------------
*-- Reposition the RP to its original position, if its
*-- safe to do so
*-----------------------------------------------------
DO CASE
CASE lnRecNo > RECCOUNT( lcAlias )
CASE lnRecNo = 0
OTHERWISE
   GO lnRecNo IN ( lcAlias )
ENDCASE
RETURN llIsAdding

*************************************************
*  FUNCTION IsAdding()
*************************************************
* Author............: Charles T. Blankenship
* Project...........: Codebook 5.0
* Created...........: 05/13/97  19:30:00
* Copyright.........: (c) Software Assets of Virginia,
*                         Inc. 1997
*)Description.......: Completely rewritten to
*)                    incorporate the additional
*)                    capability of detecting
*)                    table buffering.
* Calling Samples...:
```

```
* Parameter List....:
* Major change list.:
*****************************************************
FUNCTION IsAdding( tcAlias)

*-----------------------------------------------------
*-- Returns .T. if the user is in the midst of adding
*-- a record to the alias specified in the tcAlias
*-- parameter.
*-----------------------------------------------------
LOCAL lcGetFldState, llRetVal

DO CASE
*-----------------------------------------------------
*-- If no alias was specified ... OR ...
*-- the specified alias is not in use ... OR ...
*-- an alias that had no table buffering active was
*-- specified return a .F. since GETFLDSTATE()
*-- requires buffering to be on
*-----------------------------------------------------
  CASE EMPTY(tcAlias) OR ;
       !USED(tcAlias) OR ;
      (!EMPTY(tcAlias) AND ;
       CURSORGETPROP("BUFFERING",(tcAlias))=DB_BUFOFF)
   llRetVal = .F.
*-----------------------------------------------------
*-- CASE RECORD buffering is active, perform the
*-- original functionality of IsAdding() ...
*-----------------------------------------------------
CASE CURSORGETPROP("BUFFERING",(tcAlias)) = ;
     DB_BUFOPTRECORD OR ;
     CURSORGETPROP("BUFFERING",(tcAlias)) = ;
     DB_BUFLOCKRECORD

     llRetVal = IsAddingOriginal( tcAlias )
*-----------------------------------
*-- CASE TABLE buffering is active ...
*-----------------------------------
CASE CURSORGETPROP("BUFFERING",(tcAlias)) = ;
     DB_BUFOPTTABLE OR ;
     CURSORGETPROP("BUFFERING",(tcAlias)) = ;
     DB_BUFLOCKTABLE

     llRetVal = IsAddingTB( tcAlias )

  OTHERWISE
     llRetVal = .F.
  ENDCASE
  RETURN llRetVal

ENDFUNC

*****************************************************
```

```
*  FUNCTION IsAddingOriginal()
**************************************************
*  Author............: Paul Bienick
*  Project...........: Codebook 3.0
*  Created...........: 07/24/95  14:14:20
*  Copyright.........: (c) Flash Creative Management,
*                          Inc., 1995
*) Description.......: Returns .T. if the alias
*)                     specified in tcAlias is in the
*)                     midst of adding a new record.
*  Calling Samples...:
*  Parameter List....:
*  Major change list.:
**************************************************
FUNCTION IsAddingOriginal( tcAlias )

LOCAL lcGetFldState, ;
      llRetVal
```

```
   lcGetFldState = GETFLDSTATE(-1, tcAlias)

   IF ISNULL(lcGetFldState)
      llRetVal = .F.
   ELSE
      llRetVal = ( "3" $ lcGetFldState OR ;
                   "4" $ lcGetFldState)
   ENDIF
   RETURN llRetVal

ENDFUNC
```

IsAdding() New Functionality

This is a brand new function that is branded with the name of the original. This new function is the controlling engine for detecting whether or not a cursor is in the middle of appending a record.

The first thing to accomplish is to make sure that any one of three possible error conditions do not exist. If any of them evaluate to be true then this function should not be executed, 1) an alias was not provided (this was the original signature for the function) OR 2) the specified alias is not in use OR 3) an alias was provided AND it is not using buffering. If any of these situations exist, return a .F. to the calling program.

If an alias was provided, it is in use and it is using some type of table buffering, the next thing to determine is which type of data buffering it is use, there are two major types, record and table. Any cursor that is employing record buffering cannot have its record pointer moved without triggering an event with the database, therefore, any cursor where record buffer is active can be serviced safely by the original IsAdding function. This original code was renamed IsAddingOriginal() and is called immediately when record buffering is detected. In this situation, the basic operation of IsAdding() is exactly the same as before this change if record buffering is active.

If, however, table buffering is active, it becomes necessary to scan through each record in the cursor and test each one for an Is Adding condition. If table buffering is active in the cursor, the function responsible for handling this situation, IsAddingTB(), is called. In this function, the processing begins by first saving the original position of the record pointer. It then activates a SCAN loop that steps through each record in the cursor and looks for any record that has been appended. The test for an adding condition is performed by the original function designed for this purpose, the IsAdding() function which is now named IsAddingOriginal()

Conclusion

This modification allowed Codebook to correctly identify an Is Adding condition in all types of buffering conditions without forcing a modification to any code that previously referenced the IsAdding() function. The redesign of the IsAdding() function turned it into a switchbox that detects one of three possible conditions and performs the proper processing. The first condition is one of three possible errors that if even one exists, determine that this function should not execute. The second condition is when record buffering is active. In this case, the original functionality of IsAdding(), which now resides in the IsAddingOriginal() function, is completely capable of determining if a record is being added. The final condition occurs when table buffering is active

and each record in the cursor must be tested for an Is Adding condition. This functionality is the responsibility of the IsAddingTB() function. Now, no matter which type of buffering is active, IsAdding() can be relied upon to return the correct answer.

There is only one consideration before this solution can be safely implemented and it deals with *the* primary design assumption. Table buffered cursors are usually on the many side of a one to many relationship; therefore, the resultant set present in these cursors is almost always extremely small. This means that launching a SCAN loop while having one selected is not a major hit to performance. If however, a specific condition exists where a table buffered cursor could have a million records present, this "fix" won't seem like a fix at all … more like an anchor. So be careful, know your environment, implement good designs and this function should serve you nicely.

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: cblanke@norfolk.infi.net, www.savvysolutions.com, CompuServe 76132,2575

cDataEnvironments Described

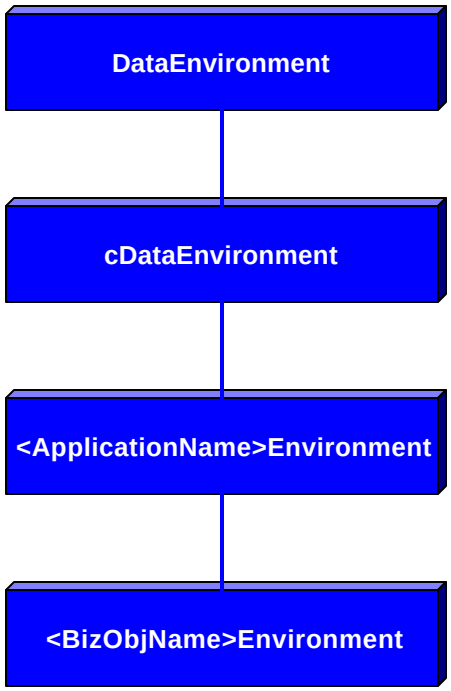
Many individuals over the years have requested a detailed explanation of the cDataEnvironment Class. Here it is in all of its glory. This article takes cDataEnvironment from its VFP roots (DataEnvironment base class) through its entire class hierarchy and will hopefully leave you much more enlightened in the end.

Charles T. Blankenship

Codebook is a difficult thing to learn but one of the more difficult tasks is mastering the hand coded data environments. The class definitions Codebook developers rely on to create data environments reside in two programs, each of which are located on the Code section of the application's project manager. The names of these two programs are ADATAENV.PRG and CDATAENV.PRG. ADATAENV.PRG contains the class definitions unique to the application being developed. CDATAENV.PRG contains the foundation class definitions from which the application specific data environments are subclassed. One of the most robust of these foundation classes is the cDataEnvironment. This article examines that class in detail.

CDATAENV.PRG - The VFP Foundation Classes

The VFP DataEnvironment base class is the foundation upon which a Codebook business object data environment is built. The exact hierarchical relationship from VFP base class (DataEnvironment) to ADATAENV.PRG business object data environment class is as follows:



A cDataEnvironment IS-A DataEnvironment.

A <ApplicationName>Environment IS - A cDataEnvironment

A <BizObjName>Environment IS-A <ApplicationName>Environment.

In a completed application, this hierarchy could look like this

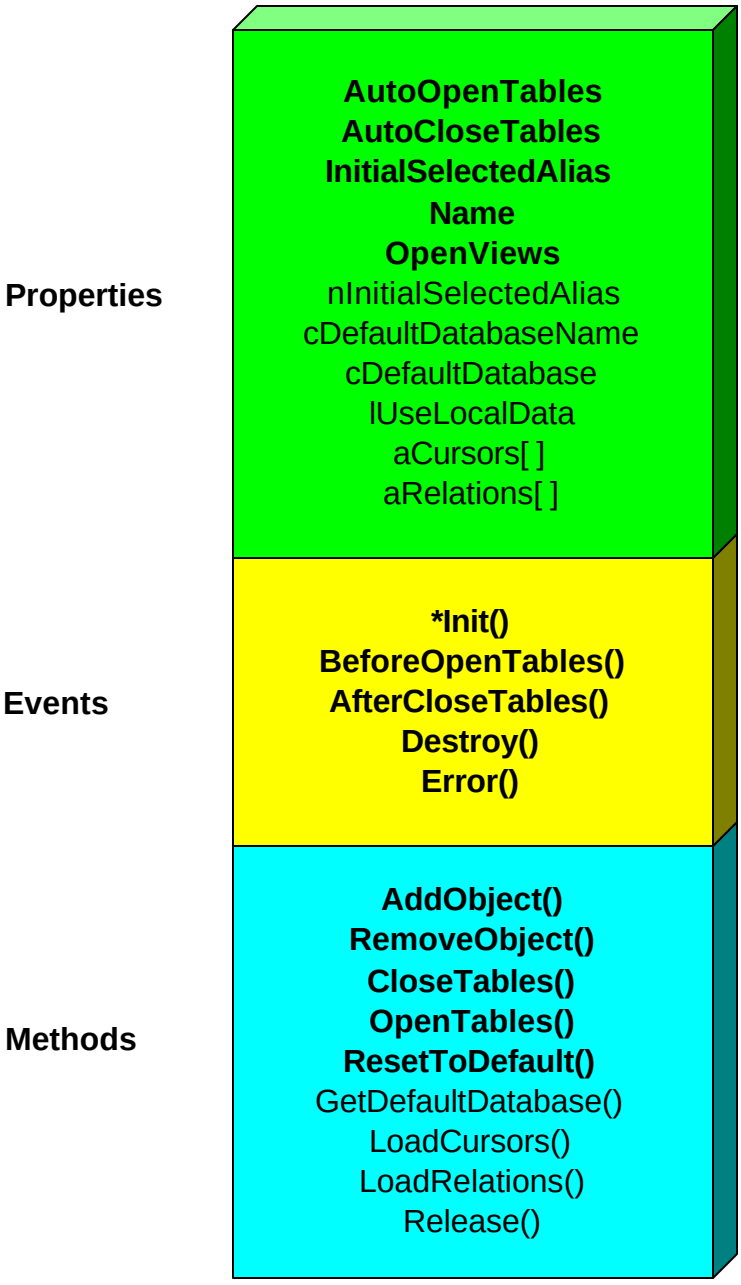
A cDataEnvironment IS-A DataEnvironment
A UserCommunicationEnvironment IS-A CDataEnvironment
A DeveloperEnvironement IS-A UserCommunication Environment

DataEnvironment Object

As per the documentation provided with VFP, The DataEnvironment "... object is a container object for the Cursor and Relation objects associated with a Form, Form Set or Report.". This means that multiple cursors and/or relations can be contained in a DataEnvironment object. This is the exact behavior desired for the data environment of a business object.

Since VFP doesn't allow these classes to be visually subclassed (very unfortunate), Flash had to subclass them programmatically to make them available for the Codebook framework. There is one serious drawback to programmatic class definitions. The properties, events and methods of the superclass are hidden from the developer unless that developer knows what those properties are. Frequently, this knowledge is not known and the lack of that knowledge results in a portion of the object's functionality being ignored, undiscovered and unexplored. Therefore, the first step in understanding the functionality of the cDataEnvironment class is to understand the PEMs of the DataEnvironment base class.

cDataEnvironment Object PEMs



DataEnvironment Properties

The following is a description of all the properties available for the VFP base class DataEnvironment. Note that these properties cannot be set at runtime without receiving an error in response. In order to set the value of one of these properties at run time the CloseTables

method must be executed before the value in the property is changed. When the OpenTables method is called, the newly established setting takes effect.

Knowing this, examine the class definition of kBizObj, eBizObj or cBizObj and look for a property named "oDataEnvironment". This property stores a reference to the business object's data environment. Also notice that it is PROTECTED. The reason this property is protected is no longer a mystery. Doing so would only invite an application programmer to generate errors since the properties of the object cannot be changed without first closing the tables, setting the property and finally reopening the tables again. This is an excellent example of using the PROTECTED classification to protect third party programmers from inadvertently hanging themselves.

AutoOpenTables - (logical) this property determines whether the cursors associated with a data environment object are loaded automatically. If set to .T., the default, the cursors are opened when the data environment object is created. If set to .F. the cursors are not opened automatically. This property is available at design time but read-only at run time. NOTE: The cursors associated with a DataEnvironment can be programmatically opened by using the OpenTables method. The ability to programmatically open the cursors in the data environment using the OpenTables method will come in handy when the behavior of the cDataEnvironment subclass is discussed.

AutoCloseTables - (logical) this property specifies whether the cursors that participate in the data environment are closed when the data environment object is released from memory. If this property is set to .T., the default, then the cursors associated with the data environment object are closed when the data environment object is released from memory. If this property is set to .F., the cursors associated with the data environment object are left open when the data environment object is released from memory. This property is available at design time but read only at run time. NOTE: The cursors can be programmatically closed by calling the CloseTables method. The ability to programmatically close the cursors in the data environment using the CloseTables method comes in handy when the behavior of the cDataEnvironment subclass is documented.

InitialSelectedAlias - (character) specifies which alias associated with a Cursor object will be set as the current alias when the data environment is loaded. This property mimics the behavior of the SELECT command.

Remember that the DataEnvironment object is a container object that can contain multiple Cursor objects. The value in this property specifies which one of those (possibly many) cursor objects is the initially selected alias when the data environment is created. Take note that this property is not directly manipulated by the developer in the cDataEnvironment class. It is indirectly set through the manipulation of a custom property in a subclass of this object, namely nInitialSelectedAlias and is used to set the value of InitialSelectedAlias in the cDataEnvironment::Init() method.

Name - (character) specifies the name used to reference an object in code.

OpenViews - (numeric) determines the type of views associated with a data environment object that are opened automatically. This property is available at design time but read only at run time. A value of 0, the default, results in both local and remote views for the data environment being

opened automatically. A value of 1 results in only local views being opened automatically. A value of 2 results in only remote views being opened automatically. A value of 3 results in no views being opened automatically. One might jump at this and say "Aha, thats how they switch between local and remote views". Don't, its not how it is done.

DataEnvironment Events

The following are the events provided by VFP for the DataEnvironment object. These things are triggered as a result of other actions that take place with the object.

BeforeOpenTables - occurs just before the tables and views associated with the data environment for a business object are opened.

AfterCloseTables - occurs after the tables or views specified in the data environment for a business object are closed. The AfterCloseTables event occurs whenever the CloseTables method is called. The Destroy event occurs after the AfterCloseTables event.

Destroy - occurs when the data environment object is released. The Destroy event for a container object triggers before the Destroy event for any of its contained objects. The container's Destroy event can refer to its contained objects before they are released.

Error - occurs when there is a run time error and allows the object to handle errors. **This event overrides the current ON ERROR routine and allows each object to trap and handle errors internally.**

The basic gist of OO error handling is that an object is responsible for handling its own kind of errors. The ones it can't handle should be passed up the object hierarchy until an object is found that knows about that type of error. Therefore, all of the VFP errors that can occur with tables, cursors, relations, etc. should have the code that handles those errors placed in a subclass of this method, namely cDataEnvironment.

There is a caveat to placing code in the Error method of a DataEnvironemnt subclass however. cDataEnvironment, uses ON ERROR to trap "errors" in its processing. If Error method code is ever written for DataEnvironment or any of its subclasses modifications will have to be made to the cDataEnvironment::CloseTables() method in order to keep it working as planned … it uses ON ERROR to trap an expected error as a matter of normal processing. Placing code in the Error() method overrides ON ERROR and breaks the CloseTables() method.

Init - occurs when the data environment object is created. This method, in a subclass of this class, provides a place where some wonderful things can be done with views, like indexing them. It is also the location where a great deal of work is accomplished for business object data environments.

DataEnvironment Methods

It is interesting to note that in the on line Help for the DataEnvironment object that the Properties and Events are adequately covered, but the Methods for the DataEnvironment were missing (in

my version anyway). They were in the Online Documentation however. This information follows:

AddObject - Adds an object to the data environment object (a container) at run time. The DataEnvironment object is a container that only accepts members whose base class is either Relation or Cursor. This method is used to create cursors and relations and add them to the business object environment in the cDataEnvironment class.

RemoveObject - Removes a specified object from the data environment object (a container) at run time

CloseTables - can be called to programmatically close the cursors associated with the data environment.

OpenTables - Programmatically opens the tables and views associated with the data environment.

ResetToDefault - Restores a property to its VFP default setting. Available at run time and design time.

Of all these methods only the ResetToDefault and RemoveObject go unused by the Codebook framework.

The Codebook subclasses of the VFP DataEnvironment make judicious use of its PEM's. In order to have an intelligent discussion of the functionality of cDataEnvironment, it was imperative to understand the functionality of its base class.

cDataEnvironment

This class definition is located in the CDATAENV.PRG and is the work horse of the business object's data environment. In order to understand Flash's cDataEnvironment, we'll examine the additional Properties and Methods added to the cDataEnvironment subclass. Secondly we'll examine how they overrode the existing properties and methods to get the desired behavior for the data environment.

cDataEnvironment Properties (custom)

Flash added six member properties, two of which are arrays to the cDataEnvironment subclass. Those properties are identified and defined below. These properties are the non-bolded ones in the illustration.

nInitialSelectedAlias (numeric) - stores the index number of the cursor in the aCursors[] array that should be initially selected when the data environment is created. This property defaults to 1 but can be overridden by the developer to point to any cursor object defined in the aCursors[] array. **NOTE:** this number is vitally important to the behavior of the business object which uses this data environment. ALL behavioral operations for a business object operate on the cursor identified as the initially selected alias. There can be many cursor objects in the data environment but only one of those is the cursor in which CRUD activities (Create, Read, Update and Delete) take place for the business object. This number identifies which cursor that will be.

cDefaultDatabaseName (character) - the name of the default database. This property is initialized in ADATAENV.PRG by the QSTART program and assumes the value provided for the database when the application is initially built. The value in this property is the filename of the database container *without* the .DBC extension (that is important as you will see later).

cDefaultDatabase (character) - This property stores the fully qualified file name of the default database. The drive, path and filename are combined to populate this property in the GetDefaultDatabase() method. The value stored in this property is constructed from information contained in the registry (the path to the database), the cDefaultDatabaseName property (the database name itself) and a default ".DBC" extension tacked on for good measure, programmatically, in the class definition.

lUseLocalData (logical) - stores whether or not the application is using local data. This is the property that Codebook uses to determine which view to use for the data environment, those prefixed with an lv_ (local views) or those prefixed with an rv_ (remote views). This property is populated in the GetDefaultDatabase() method.

aCursors[ ] (character array) - this array stores the names of the cursor objects that are defined by the developer to be a part of the business object's data environment. The actual values to place in this array are defined by the developer in a subclass of cDataEnvironment in the ADATAENV.PRG

aRelations[ ] (character array) - this array stores the names of the relation objects that are defined by the developer to be a part of the business object's data environment. The actual values to place in this array are defined by the developer in a subclass of cDataEnvironment in the ADATAENV.PRG

cDataEnvironmentProperties (overridden)

Flash also overrode several of the base class properties in this class definition as well. The information placed in those properties is identified and described below.

AutoOpenTables (logical) - overridden in this class definition to ensure that the cursors and relations associated with a business object are automatically opened when the business object is created. This is why Codebook developers do not have to worry with opening tables and views in their data environment.

AutoCloseTables (logical) - overridden in this class definition to ensure that the cursors and relations associated with a business object are automatically closed when the business object is released. This is why a Codebook developer does not have to worry about closing views and tables when the business object goes out of scope. Setting this property to .T. forces this to happen automatically.

Name (character) - overridden in this class definition so the name of the data environment is "cDataEnvironment". To verify this, make the oDataEnvironment property in the xBizObj PUBLIC, run the application and poke around in DEBUG for the Name property of the

oDataEnvironment object. You'll see it is "cDataEnvironment" *When completed, set the classification for this property back to PROTECTED so as to prevent inadvertent errors.*

CDataEnvironment Methods

The following methods define the behavior of the cDataEnvironment class. When reading the following paragraphs please look at a copy of cDataEnvironment code. This will help in understand these methods.

Init - This method accomplishes the following things:

1 Ensures that the object being created from this class definition has been subclassed at least once (cDataEnvironment is an abstract class definition the reason for is explained later in the article)
2 Gets the name of the application's default database and populates the cDefaultDatabase property with the fully qualified database container name (drive, path and filename) and validates this information. Also encapsulated within this method is where the business object determines whether or not the user wants to use local or remote data
3 Loads the Cursors that the developer has defined for a business object's data environment (a subclass of this object) into the aCursors[] member array
4 Adds the defined cursors to the data environment.
5 Assigns the name of a cursor in the aCursors[ ] array to the InitialSelectedAlias (character) property by using the nInitialSelectedAlias to identify which element of the array should be used selected as the default cursor
6 Loads the Relations that the developer has defined for a business object's data environment
7 Adds the defined relations to the data environment.
8 Opens the tables if the AutoOpenTables property is set to .T. by calling the THIS.OpenTables() method if requested to do so.

Notice that the Init method makes a call up the class hierarchy to its subclass' LoadCursors method so it can identify the actual names of the cursors to create. It then adds those cursors to the DataEnvironment object via the AddObject method of the VFP DataEnvironment base class. (Just a bit of trivia, the name of the cursor object contained in the data environment container is a variation of the cursor class definition. If the cursor class definition is "v_Developer" the name of the cursor object, once it is created, is "ov_Developer"). It is interesting to notice that when the cursor objects are created, they are not yet present in the business object's data environment. They only present themselves in the data environment after OpenTables method is called, which, of course, is the last action to take place in the cDataEnvironment::Init() method.

Also notice that Flash did not use the scope resolution operator :: to directly call the DataEnvironment::OpenTables() method from the Init method. They sent this call all the way up the class hierarchy of the data environment and forced it to eat its way back down. The impact of this is that you can override the subclass' OpenTables method and define pre- as well as post-processing for the action of opening tables. Very nice flexibility that is not immediately evident by viewing the templates provided in the ADATAENV.PRG. The reason these methods are not imminently noticeable is that the data environment classes are programmatically created. While all of the PEMs of the superclasses are *available* for use, programmers that do not know the structure of the underlying superclasses cannot see them in this environment like they can in a visual environment. This is why a redefinition of the business object data environment class that

includes each and every method available at that level is helpful to beginning developers. It illustrates what functions are available and hopefully piques some interest as to how they can be used.

GetDefaultDatabase - this method first determines if the application object exists. If the application object exists it then checks to see if the default database name has been defined for the application. By default, if the ADATAENV.PRG is intact and the QSTART.APP worked correctly, the name provided for the database in QSTART.APP is defined at the top of the ADATAENV.PRG. That code looks similar to the following (each application is different in content due to different values entered into QSTART.APP by the developers).

```
DEFINE CLASS LBSEnvironment AS CDataEnvironment
       cDefaultDatabaseName = 'LBS'
ENDDEFINE
```

Once the name of the default database is known, Codebook then is ready to determine whether or not it should use local or remote data. It does this by asking the application object to get the setting stored for the Use Local Data question. This question is "asked" on the User Preference form. This information is stored as a YES or NO in the INI or registry and is translated into a .T. or .F. respectively by code in this method. The result is placed in the lUseLocalData property. The value in this property, at a later time, determines which views get used, those that begin with an "lv_" or those that begin with an "rv_".

The desire to use local data is trapped by the User Preferences screen, stored in either the registry or the INI file and the value stored there determines whether or not local or remote views are used. In my registry, this information is stored in HKEY_CURRENT_USER | Software | Lathrop's Business Systems | LBS Operations Management Application | 1.0 | Data Settings.

Another action that occurs here is the population of the lcKey variable. This variable stores the key used to identify, in either the registry or an INI file, the section that stores the location of the default database (drive and path). For the application I am using to write this article, the value in lcKey is "DBC Locations - Local' and is the exact name of the following section in my registry. HKEY_CURRENT_USER | Software | Lathrops Business Systems | LBS Operations Management Application | 1.0 | DBC Locations - Local.

The next task is to extract the actual value stored in the DBC Locations - Local section. To do this, another call to the goApp.SystemSetting(), using the value in lcKey ("DBC Locations - Local"), asks the application to cough up the location of the default database. The return value of this method is placed in the lcDefaultDatabase variable and contains something like along these lines "D:\LBS\DATA\".

The final objective of the GetDefaultDatabase() method, and the one that determined its name, is to construct the fully qualified default database name by concatenating the path of the application (provided by you in QSTART with a "\DATA\" tacked on and subsequently stored in the INI file or the registry), database name (provided by you in QSTART) and finally the .DBC extension (added by this method since it is the default extension for a database file). This method then returns something like "D:\LBS\DATA\LBS.DBC".

This method did quite a bit. It first determined whether or not the user wanted to use local or remote data and populated the lUseLocalData property with the appropriate value. It then determined where the location of the default database is stored in the Registry (or INI file). It then extracted the location information for the default database out of that location and used it, coupled with the default database name to construct the fully qualified name of the .DBC to use. Whew!

LoadCursors - This method is simply a place holder (abstract method technically speaking) that provides a location for the developer to define the exact names of the cursor class definitions that make up the data environment of the business object. This method is empty at this level and is meant to be overridden in ADATAENV.PRG by the developer. The code in a business object's data environment class definition for this method can look something like this.

```
DEFINE CLASS LocationEnvironment AS LBSEnvironment
   FUNCTION LoadCursors()
      DIMENSION this.aCursors[1]
          this.aCursors[1] = 'v_Location'
   ENDFUNC
ENDDEFINE
```

The call to LoadCursors in the Init method executes the above code which populates the aCursors[] array. The information in the aCursors array is then used by the AddObject method to create the developer specified cursors.

LoadRelations - This method, like the LoadCursors method, is simply a place holder that provides a location for the developer to define the exact names of the relation objects that make up the data environment of the business object. This method is empty at this level as well and is meant to be overridden in the ADATAENV.PRG

AddObject - This method of the DataEnvironment VFP base class is called from the cDataEnvironment::Init() method and is used to add the cursor and relation objects defined in the subclassed LoadCursors() or LoadRelations() methods to the business object's data environment.

OpenTables - This method calls the DataEnvironment::OpenTables method whose primary function is to create the views, tables and relations in the data environment. Notice that in the cDataEnvironment::Init() method, that the cursor and relation objects are added to the data environment *before* the call to the open OpenTables method. Suspending operations before the call to OpenTables reveals that those cursors don't really exist in a work area, yet. The cursors / relations are not *created* until after the OpenTables method successfully completes its task.

The next action of OpenTables is rather interesting. Notice that the NODEFAULT command is issued. The reason for this is that the OpenTables method is called manually. Remember the default behavior of the open tables property? Did you notice that the property was set to .T. in the header portion of the cDataEnvironment class definition? This means that the default behavior of the VFP DataEnvironment base class is to automatically open tables when the data environment object is created. However, Codebook, does this by calling DataEnvironment::OpenTables programmatically. The NODEFAULT command overrides the default behavior and prevents OpenTables from occurring twice, once because the method was called manually and the second time through default behavior of the base class. The NODEFAULT command stops the default behavior of the method in which it is issued.

The final action of OpenTables is even more interesting. Notice that a command is being built that determines the name of the database to which to set as a default. This information is obtained from the ***cursor object designated to be the initially selected alias.*** The structure of this command in the application I am using looks like this:

```
SET DATABASE TO THIS.ov_Location.Database
```

Remember when I said that the initially selected alias is the primary CRUD work area of the business object? Well, cursors know several things. One of those things is the name of the database to which they belong (this is not as automatic and smart as it sounds. This information is determined by the cursor, when it is created, from its parent via the cDataEnvironment.cDefaultDatabaseName property from *this* class definition). This is how Codebook ensures that the proper database is set when the business object's data environment is instantiated. This will be the topic of a future article, Multiple Database Operations in Codebook.

CloseTables - this method simply calls the DataEnvironment::CloseTables method which does just that, closes the tables in the data environment.

Release - this method is added to all Codebook class definitions in order to provide a standard method for removing an object from memory. The release method calls the release COMMAND and passes a reference of the data environment object to it. This command removes the data environment from memory. Doing this then triggers the Destroy Event.

Destroy - this method first checks the setting of the AutoCloseTables property of the data environment object. If this property is set to .T. then the CloseTables method is invoked. Also notice that the call to CloseTables uses THIS.CloseTables instead of DataEnvironment::CloseTables. This has the same ramifications as it did for OpenTables. This enables the developer to provide pre- and/or post- processing at the subclass level for the process of closing tables.

cDataEnvironment Conclusion

The cDataEnvironment class definition is responsible for ensuring an object cannot be created directly from this particular class definition, retrieving the fully qualified name of the default database, determining if the user wants to use local or remote data, creating and loading the cursors and relations defined by the developer as being a part of the data environment, setting the default database to the database of the cursor in the initially selected alias of the business object and automatically opening and closing the cursors in the data environment if instructed to do so. I told you it was a work horse.

ADATAENV.PRG Explored

Now that the behavior of the cDataEnvironment super class is understood we are much better equipped to understand the application specific class definitions in the ADATAENV.PRG. The first one we run into in ADATAENV.PRG is as follows:

```
DEFINE CLASS LBSEnvironment AS CDataEnvironment
      cDefaultDatabaseName = 'LBS'
ENDDEFINE
```

This class definition overrides the cDefaultDatabaseName property of the cDataEnvironment class. We know that the purpose of this property is twofold. First, it aids in the creation of the fully qualified name of the application's default database. The path for this database is extracted from the registry or the INI file, the information in this property provides the name of the database file stem and to complete the fully qualified database name, the .DBC

extension is tacked on programmatically. Secondly, this property is used by the cursors when they are being created in order to determine the database to which they belong.

```
DEFINE CLASS LBSSalesEnvironment AS CDataEnvironment
       cDefaultDatabaseName = 'LBSSALES'
ENDDEFINE
```

The next class definition in ADATAENV.PRG is the one used to set up the reporting environments. An example of this follows:

```
DEFINE CLASS LBSReportEnvironment AS LBSEnvironment
       AutoCloseTables = .F.
ENDDEFINE
```

This subclass definition simply overrides the default value of the AutoCloseTables property of the cDataEnvironment class with a value of .F. Setting this property to .F. causes the call to CloseTables in the cDataEnvironment::Destroy() method to be skipped, which in turn ensures these cursors will not be closed automatically when the data environment object goes out of scope. Therefore, subclassing all reporting environments from the LBSReportEnvironment (in this application) guarantees that the tables and views of the data environment will not be closed when the creating data environment object is released from memory.

Finally, notice that the data environment classes used to create data environments for business objects inherit from the LBSEnvironment (this name will be different for each ADATAENV.PRG … these names are created from the information you provided when you ran QSTART). A detailed discussion of these class definitions will take place once the full functionality of the Cursor and Relation objects and their subclasses are explored.
The following is a modified template for a business environment

```
*-------------------------------------------------
*-- Template definition for a standard data
*-- environment used by a business object.
*-------------------------------------------------
*DEFINE CLASS <BizObjName>Environment AS ;
*       LBSEnvironment
*
*  nInitialSelectedAlias = 1
*
*  FUNCTION Init()
*    cDataEnvironment::Init()
*  ENDFUNC
*  FUNCTION LoadCursors()
*    DIMENSION this.aCursors[<NumberOfCursors>]
*    this.aCursors[1] = '<v_ViewClassName>'
*    this.aCursors[2] = '<cLocalTableClassName>'
*  ENDFUNC
*  FUNCTION LoadRelations()
*     DIMENSION this.aRelations[<NumberOfRelations>]
*     this.aRelations[1] = '<cRelationName>'
*  ENDFUNC
*  FUNCTION OpenTables()
*     cDataEnvironment::OpenTables()
*  ENDFUNC
*  FUNCTION CloseTables()
*     cDataEnvironment::CloseTables()
*  ENDFUNC
*  FUNCTION BeforeOpenTables()
*  ENDFUNC
*  FUNCTION AfterCloseTables()
*  ENDFUNC
*ENDDEFINE
```

This expanded template definition serves as a reminder that some additional options are available when creating data environments. Remove all methods that are not used once the actual data environment class definition is complete to keep ADATAENV.PRG from becoming too cluttered.

Conclusion

Dissecting this class has provided a foundation upon which to fully understand Codebook business objects. This class definition is the foundation of the data environment for every Codebook business object. However, the journey is only half over. Data environments have HAS-A relationships with Cursors and Relations. Therefore, in the following articles, the collaborating objects used by the cDataEnvironment completes the understanding of Codebook's programmatic data environments. These collaborating objects include Cursor, cCursor, cTableCursor, cFreeTableCursor, cDynamicViewCursor, cReportDynamicViewCursor and finally cRelation.

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: cblanke@norfolk.infi.net, www.savvysolutions.com, CompuServe 76132,2575

cCursors And Their ILK

Cursors make up a significant component of a business object's data environment. Understanding their behavior is critical to understanding how local and remote views can coexist peacefully in the same Codebook application.

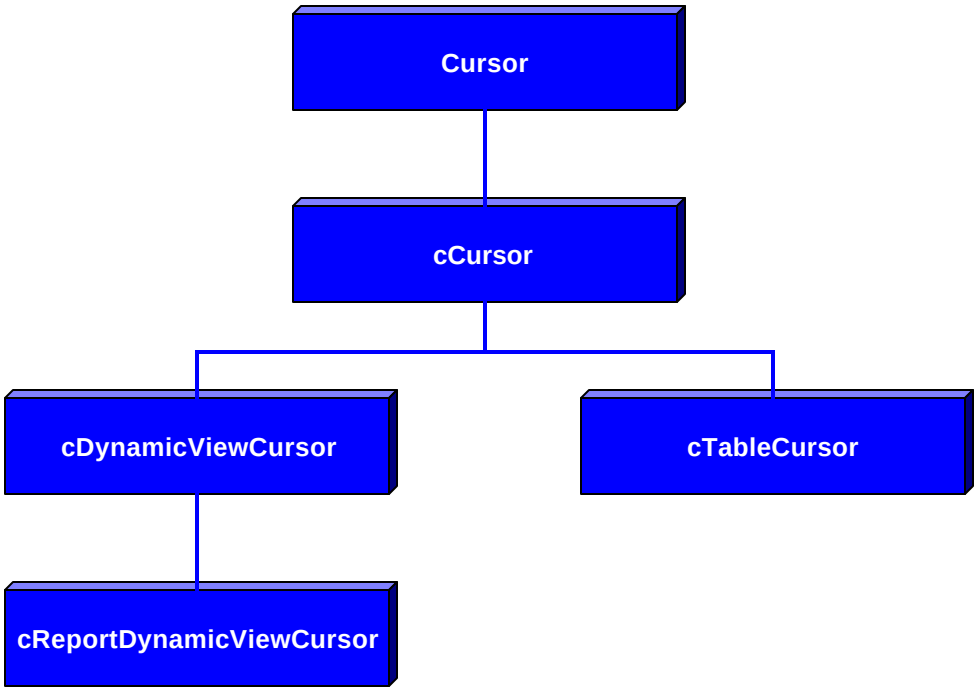
Charles T. Blankenship

Introduction

Before a class can be fully understood, it is important to understand its heritage. In Codebook, the heritage of the Cursor objects is as follows:

A cCursor IS-A Cursor
A cTableCursor IS-A cCursor
A cDynamicViewCursor IS-A cCursor
A cReportDynamicViewCursor IS-A cDynamicViewCursor

Visually, this class hierarchy looks like this:

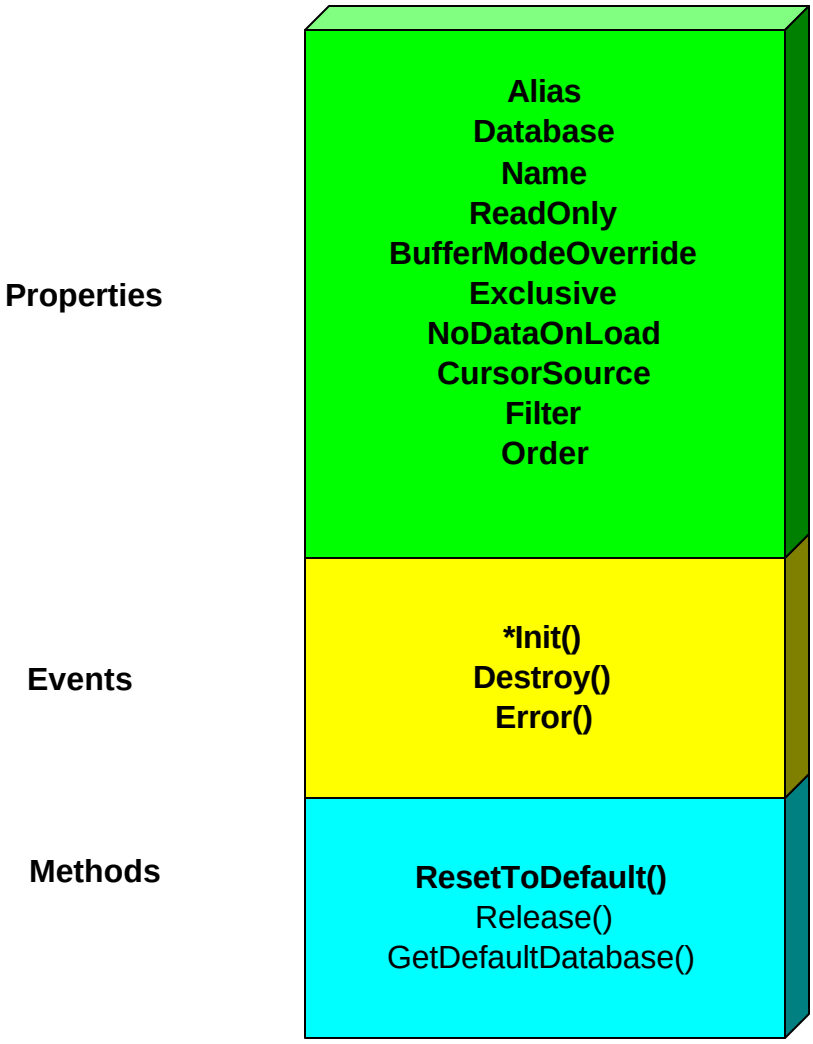


Therefore, just as the DataEnvironment class had to be understood before we could discuss the cDataEnvironment class, the VFP Cursor base class must be understood before an intelligent discussion can be undertaken about any of its subclasses.

The cursor object is VFP's primary means of associating our long loved table with an object in the OO world. The best way to begin to understand cursors is to first understand all of their properties, events and methods and then begin to associate the new concepts with our old familiar ones. The properties of the Cursor object and their related descriptions follow. With very

few exceptions, the information provided is exactly as it appears in the on-line help. Note that setting a cursor object property at run time generates an error *(with the exception of the Filter and Order properties, which can be set at run time)*. For a new property setting to take effect, the CloseTables method must be called first, only then can the property's value be set. When the OpenTables method is issued, the cursor is created again with the new settings.

cCursor Object PEMs



cCursor Properties

Alias (character) - Specifies the alias used for each table or view associated with a Cursor object. Available at design time and run time. When the data environment is created, each table or view associated with a Cursor object is assigned an alias. *By default, the alias name is the*

same as the name of the table or view. When a value is placed in the Alias property, this value overrides the default alias name (this information becomes important later). The Alias property mimics the behavior of the USE command's ALIAS clause.

Database (character) - Specifies the path to the database that contains the table or view associated with the Cursor object. Read-only at design time; read-write at run time.

Name (character) - the name used to reference the cursor object in code.

ReadOnly (logical) - specifies whether a table or view associated with a Cursor object allows updates. Available at design time; read-write at run time. This property mimics the NOUPDATE clause of USE.

BufferModeOverride (numeric) - specifies the type of buffering that controls the cursors behavior. The following table provides the descriptions, Codebook constants and numeric values the associated with this property

Buffering Description	Constant	Numeric Value
No Buffering	DB_BUFOFF	0
Pessimistic Row Buffering	DB_BUFLOCKRECORD	2
Optimistic Row Buffering	DB_BUFOPTRECORD	3
Pessimistic Table Buffering	DB_BUFLOCKTABLE	4
Optimistic Table Buffering	DB_BUFOPTTABLE	5

Exclusive (logical) - specifies whether a table associated with a Cursor object is opened exclusively. Available at design time; read-write at run time. If this property is set to .T., then the table associated with the cursor cannot be used by another in a multi-user environment. If this property is set to .F. then the table associated with the view will be available for use by others in a multi-user environment. The Exclusive property mimics the behavior of the USE command's EXCLUSIVE and SHARE clauses.

NoDataOnLoad (logical) - Causes the view associated with a Cursor to activate without downloading data. Available at design time; read-write at run time. If this property is set to .T. then the view is opened with no attempt to download records from the data source. If this property is set to .F. then the view associated with the cursor is opened with all available data. The NoDataOnLoad property mimics the behavior of the NODATA clause of USE.

CursorSource (character) - Specifies the name of the table or view associated with a Cursor object. Read-only at design time; read-write at run time. For views, this property specifies the name of the view in a database. For tables in a database it specifies the long table name. For free tables it specifies the full path to the free table.

Filter (character) - Excludes records that do not meet the criteria in the specified expression. Available at design time; read-write at run time. This property can be set to any Visual FoxPro expression that evaluates to a .T. or .F. Typically they are expressions that operate on a set of records. *This is one of two properties that can be set at run-time without causing an error condition to occur.* This property mimics the behavior of the SET FILTER command.

Order (character) - Specifies the controlling index tag for a Cursor object. Available at design time and run time. Use the Order property to specify the order in which records are displayed or accessed. The Order property mimics the behavior of SET ORDER.

Importance of Association

Did you notice that almost every property associated with a Cursor object has an equivalent FoxPro command that we are familiar with from our pre-OOP days? Simply think of a cursor as an object that has many of the capabilities of our old friend the USE command (with a couple of extras thrown in, namely SET FILTER and SET ORDER). This functionality has simply been migrated into an OOP environment via a VFP base class called Cursor.

Events

The following are the events associated with the cursor object. Since these events are common in behavior to the events of every other object, there is little need to discuss them here, Init, Error, and Destroy. Init occurs when the object is created. Destroy occurs when the object is released from memory. Error occurs when something performed an action it shouldn't have.

Method

The only *method* associated with a Cursor object, ResetToDefault is not very useful at all in this environment. If I think of a use later on, I'll be sure to let you know … but don't hold your breath.

Responsibilities of the cCursor Class

(Please reference the code while reading the following description)

This class is the first level of abstraction from VFP's Cursor base class. Populating its properties with default values ensures that every Codebook cursor is, by default, governed by Optimistic Record Buffering (BufferModeOverride property), contains no data when it is loaded (NoDataOnLoad property), has no filter condition in place (Filter property), has no order set (Order property) and is updateable (ReadOnly property).

Its second purpose is to ensure that no object can be directly instantiated from cCursor. Every time the IsAbstract() method is used in an Init() method of a class definition in Codebook, it is there to prevent an incomplete class definition from being used directly to create an object. This is the hallmark trait of an abstract class. An abstract class is a class that has something missing, i.e. this class must be subclassed and additional information must be provided before the object is completely defined. Can you pick out what is missing from this class definition that qualifies it as an abstract class? Look at the properties that are provided with the VFP Cursor base class and look at the properties that are overridden in this class definition. The very important piece of missing information is .CursorSource. At this level of abstraction, there is no table or view associated with this object. Therefore, this class definition cannot be used to create an object, it has no data source specified

The next responsibility of this class is to determine what database this view or table belongs to. This is accomplished simply by asking its parent to cough up the information. Notice that this is not a method of determining "true" ownership. There is no checking to make sure that this table or view belongs to the specified database, it only blindly asks its parent to which

database it belongs. The owning database is discovered by a call to a member method named GetDatabaseName when the object is instantiating.

Take a look at cCursor::GetDatabaseName() you will find the second reason this class definition is abstract. Notice that there is an unqualified call to THIS.PARENT. First of all, that parent had better exist and secondly, that parent better have a property named cDefaultDatabase. Therefore, not only isn't the CursorSource defined at this level, the underlying assumption during the creation of this object is that it is contained in another object. That containing object is cDataEnvironment and its subclasses. Quickly refer to the Init( ) method of the cDataEnvironment class. Notice that the database is defined *before* the ADDOBJECT() method is called to create the cursors. Now you know why the database is defined before the cursors are created, the Cursors refer to their parent in order to find out to which database they belong and the parent, cDataEnvironment, doesn't know herself until the cDataEnvironment::GetDefaultDatabase() method is executed.

Also notice a very important point. This behavior exhibited by cCursor of blindly getting the database to which the cursor belongs from the cDataEnvironment seems to indicate that every cursor belonging to a data environment is assumed to belong to the same database.

Justification for creating an Abstract Superclass

One excellent reason for creating an abstract superclass is to define common behavior required by several divergent subclasses. What I mean by divergent is this. Examine the cTableCursor and cDynamicViewCursor class definitions. Each one of them inherit from the cCursor class in order to receive the same behavior and characteristics however, that is where the similarities end.

What Flash did was look at the required properties and behavior of tables contained in a database and views. They determined that both of these animals can be governed by buffering, can have a filter and an order set, can be opened with or without data (initially), can be specified as read only and finally, both can be associated with a database. By defining the cCursor class they have provided a common place where these properties and behaviors can be stored in one place and used by two divergent limbs of the class hierarchy. The creation of abstract classes decreases maintenance by reducing the amount of code to maintain to one copy.

The cCursor class definition is a textbook illustration of how an abstract class should be used. If you have two classes that have similar properties and methods in both class definitions, consider creating a class to house those common properties and methods. You can subclass to provide the additional functionality needed to make a fully functioning object and you have reduced your maintenance burden. Booch refers to this a "promoting" behaviors and properties up the class hierarchy. This also follows an OO rule: Code it once and use it many times. If similar functioning code is being duplicated throughout an application the only thing that is increasing is the maintenance burden if that task's behavior must change.

The cDynamicViewCursor Explained

At this point, the family tree forks, one limb describes the behavior for tables that are part of a database and the other limb describes the behavior for the views that are part of the database. The remainder of this article concerns itself with the cDynamicViewCursor.

The cDynamicViewCursor is subclassed from cCursor and therefore inherits all of that class' properties, events and methods. This means that views can be filtered, specified to contain data upon loading and/or be read only at our request. This type of thing is easily forgotten in the ADATAENV.PRG since none of these properties are specifically identified at that level (this is

why I reprinted the manual). You have to know the structure of the underlying class hierarchy in a programmatic environment to know the true capabilities of the classes defined in this manner.

Examine the cDynamicViewCursor class definition during the remainder of this article. First notice that the NoDataOnLoad property is overridden to ensure no data is loaded when the cursor is created. This was to ensure that the forms instantiated quicker. The initial design assumption was that people would use the cBizObjMaintForm where view parameters are populated and the view populated with data when the Execute button was pressed by the user. Second, notice that two protected member properties are defined for this class definition. The first is lUseLocalData and the second is cCursorSource. These two properties are instrumental in being able to use remote or local views on user demand.

CDyanmicViewCursor::Init() method

As far as methods go, the only enhancement at this level is some pre-processing performed before the call to the superclass Init method, cCursor::Init( ). The first thing that takes place in this method is to ensure, once again, that this class definition cannot be used to directly instantiate an object. Something is missing that prevents this class definition from creating a complete object. In this case, just as in cCursor, the missing participant is the PARENT. The other missing component at the cCursor level, CursorSource, is taken into account in the immediately following code.

The next step is to determine whether or not this application is using local data or remote data. To do this, the cDynamicViewCursor class simply asks its parent, cDataEnvironment, "Are you using local or remote data?". (If you turn your speakers up *really* loud, you can hear this when you instantiate each business object). Take a moment to glance at the cDataEnvironment::Init() method. Notice that the cDataEnvironment::GetDataBase() method is called *before* the cursors are created. Remember, that method not only determined the default database, it was also responsible for asking the application which data source the user chose, local or remote.

The next step is to derive the name of the control source. Codebook provides two possible sources for this information. The most common is from the ADATAENV.PRG where the cControlSource property is defined by the developer. The following class definition was extracted from a functioning ADATAENV.PRG.

```
DEFINE CLASS v_IDSMailSettings AS CDynamicViewCursor
       NoDataOnLoad = .F.
       cCursorSource = 'v_IDSMailSettings'
ENDDEFINE
```

When the developer creates this class definition in ADATAENV.PRG they are providing the *root* name of the control source. This *root* is derived by chopping the first letter off of the view to be used. In this case, lv_IDSMailSettings becomes v_IDSMailSettings … this also means that rv_IDSMailSettings becomes v_IDSMailSettings.

The next step in the Init method's processing is to save the Alias the developer has sepcified they want to use when referencing the view or table. Notice that if there is a value in the Alias property, then that value is used. If not, then the value present in the cCursorSource property is used. The reason this IF statement exists will become very clear shortly.

Next, the point arrives where the lUseLocalData value is translated into an "l" or an "r", tacked on to the beginning of the root view name and slapped into the ControlSource property. This is how the lv_IDSMailSettings view becomes the data source when the user specifies to use

local data and the rv_IDSMailSettings becomes the data source when the user specifies they want to use remote data.

If the code stopped here we would have a serious problem when trying to specifying the ControlSource values for text boxes, edit regions, etc. on the business objects. Do you remember the definition for the Alias property at the beginning of this article? It stated the following "When the data environment is created, each table or view associated with a Cursor object is assigned an alias that is, by default, the same as the table or view name". If no further processing took place, the controls on the business objects would have to have rv_IDSMailSettings.FieldName when remote views were being used and lv_IDSMailSettings.FieldName when local views are being used. If you have ever created a Codebook business object, you know that this is not necessary. The only thing required is the v_IDSMailSettings root of the view name. This is really nice since we do not have to change the value of the ControlSource property for every control when the user switches from local to remote data.

The next step is to ensure that each cursor, whether it be associated with a remote or local view, can be referenced by the same alias. We just happen to have an Alias property to work with that "mimics the behavior of the USE command's ALIAS clause". Several steps ago, either the Alias or the root name of the view was saved off to later use. It now comes into play. That saved value is now assigned to the Alias property and overrides the default values of rv_<ViewName> or lv_<ViewName>. This enables a reference to both rv_IDSMailSettings and lv_IDSMailSettings to be accomplished using the same alias v_IDSMailSettings.

The final act in the cDynamicViewCursor is to call the superclass functionality by calling cCursor::Init( ). We now know that the only thing this call performs is to assign the name of the database with which this view is associated to the DataBase property of the view. At this point, the cursor is ready for operation. It is only waiting for the OpenTable command before it can launch into service of our business object. This is issued in the cDataEnvironment::Init() method.

Conclusion

There are three basic purposes of the cDynamicViewCursor class. The first is to ensure the view does not try to pull data from the data source when it is first created, Second, it is responsible for deriving the full name of the view based upon the root view name provided by the developer in the ADATAENV.PRG and the user preference for data source provided on the User Preference form. It then assigns that derived value into the ControlSource property. Finally it overrides the default value of the Alias property, which was either rv_IDSMailSettings if a remote view is being used or lv_IDSMailSettings if a local view is being used, with the root view name, v_IDSMailSettings, so we can reference remote and local views via the same alias.

I hope this little discussion has removed some of the mysticism surrounding the remote and local view capability of Codebook. Every time I thoroughly dissect a Codebook class, I marvel at the simplicity and functionality created by Flash. How we ever got this code for $40.00 is a mystery to me. We got *very* lucky.

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: cblanke@norfolk.infi.net, www.savvysolutions.com, CompuServe 76132,2575

The Age Old Table

The age of the table, even in the world of Client/Server computing is by no means over, they still have many uses.

Charles T. Blankenship

Notice that there are two cursor based classes created to be used with tables (and not views), the cFreeTableCursor and the cTableCursor. The cTableCursor is meant to be used with tables that are part of a database and the cFreeTableCursor is meant to be used with a table that is not part of a database. This article explains each of these class definitions.

The cTableCursor (associated with a database)

The first responsibility of the cTableCursor is to directly populate the alias property with the name of the table (provided when the following code is created in the ADATAENV.PRG by the developer) as soon as the Init() method executes. The purpose of this is to ensure any relationship objects that are added to the data environment are properly set up.

```
*--------------------------------------------
* Cursors that represent tables in a DBC
*--------------------------------------------
*-- Template definition for a cursor used to
*-- represent a table that is attached to a DBC.
*--------------------------------------------
* DEFINE CLASS <ClassName> AS CTableCursor
*      CursorSource = '<TableName>'
* ENDDEFINE
```

Secondly, examine the parent class of the cTableCursor and notice that it is subclassed from cCursor. From the previous article we know that the primary function of the cCursor class definition is to get the name of the database with which the cursor is associated, from the parental data environment object, and assign that value to the Database property of the cursor object being created.

As far as cTableCursors go, those two responsibilities about cover it all. Remember, that with table cursors all of the VFP base class properties and events are available as well (Alias, Order, Filter, ReadOnly, Exclusive, NoDataOnLoad, BufferModeOverride, etc.) Forgetting about these properties in a programmatic environment is an easy mistake to make. Also notice that cTableCursor is not programmed as an abstract class (the call to IsAbstract is not present in the Init method). However, notice that two pieces of critical information are still missing that prevent this class definition from being used to directly instantiate an object, the CursorSource and a parent, cDataEnvironment. Therefore, I recommend placing the following code at the beginning of the Init method in the cTableCursor class definition.

```
IF IsAbstract(this.Class, "CTableCursor")
   RETURN .F.
ENDIF
```

cFreeTableCursor (no database association)

The fact that a cFreeTableCursor is not associated with a database is immediately apparent as soon as its parent is identified. Its parent is the venerable workhorse and VFP baseclass, the Cursor.

The functionality provided by the cFreeTableCursor is first to assure that this class definition cannot be used to directly instantiate an object. The second is to bring this class in line with all other Codebook classes by giving it the method common to all Codebook objects to release them from memory, the Release method. The sole function of the Release method is to use the RELEASE command and pass it a reference to the current object by using the THIS keyword. (I only wish cDataEnvironment was as easy to describe.)

Using the Table Cursors

An excellent use of table cursors in a client server environment is in the capacity of a local lookup table. An combination of views, tables and finally free tables can be mixed and matched as desired in the same data environment.

The Creation of a Business Object Data Environment

Now that the nature of cDataEnvironment and the cursor base classes are understood, lets take a look at the instantiation of the Codebook business object data environment.

Charles T. Blankenship

The following code defines all of the objects required to instantiate the data environment for the Inventory Item Business Object for Lathrop's Business Systems.

```
*--------------------------------------------------
*--        Inventory Item Business Object          --
*--------------------------------------------------
DEFINE CLASS LBSEnvironment AS CDataEnvironment
       cDefaultDatabaseName = 'LBS'
ENDDEFINE

DEFINE CLASS InvItemEnvironment AS LBSEnvironment

nInitialSelectedAlias = 1

FUNCTION LoadCursors()
   DIMENSION this.aCursors[2]
      this.aCursors[1] = 'v_InventoryItem'
 this.aCursors[2] = 'InvType'
   ENDFUNC
ENDDEFINE

DEFINE CLASS v_InventoryItem AS CDynamicViewCursor
   cCursorSource = 'v_InventoryItem'
ENDDEFINE

DEFINE CLASS InvType AS CTableCursor
  CursorSource = "InvType"
  Order        = "Primary"
ENDDEFINE
```

Associating the data environment class definition to the business object

The first step to associate the InvItemEnvironment class definition to the business object. This is accomplished by placing **"InvItemEnvironment"** in the cDataEnvironment property of the oDELoader, an object contained within the buisness object itself. When the business object is instantiated, one of the first things it does is create the contained oDELoader object which in turn references this class definition for instructions on what this data environment is really all about. The following paragraphs detail what happens when a data environment object is created.

When oDELoader is created, processing begins in the Init method in the InvItemEnvironment class definition. Since there is no Init code for the InvItemEnvironment class definition, program

execution flows through the class hierarchy to the LBSEnvironment. Since there is no Init code at this level either, program execution falls through to the cDataEnvironment. Notice that pre- and post-Init processing can be placed in any of these class definitions at will. The absence of an Init method in the InvItemEnvironment and the LBSEnvironment means only that no use for this processing is needed by default. It is always available if a developer discovers a need.

Am I abstract?

In the cDataEnvironment Init method the first thing to check for is if an object is being created directly from the cDataEnvironment class. Since this processing was initiated by the class named InvItemEnvironment the IsAbstract function returns a .F. This means that the object being created is not an abstract object but is based upon a subclass where the missing properties or behaviors have "supposedly" been provided by the developer.

What's my database's name?

The next step is to retrieve the name of the default database. If the application object exists and the file name of the default database has been provided (examine the LBSEnvironment class definition to see where this information comes from) determine if the user wants to use local data or not. If they want to use local data, determine where the path name of the local database is stored and set the lUseLocalData property to .T. If they do not want to use local data, determine where the path name of the remote database is stored and set the lUseLocalData property to .F. Note that the defined constant DBFLOCATION_LOCALKEY references information stored in the APPINCL.DBF and the default value for this constant is "DBC Locations - Local".
Next, retrieve the true path of the database being used (local or remote) from the INI or Registry and add a backslash to the pathname if necessary.

Finally, create the fully qualified file name for the desired database by concatenating the drive and path information, pulled out of the INI or Registry, the name of the database, retrieved from the cDefaultDatabaseName property in the LBSEnvironment class definition located in the ADATAENV.PRG and finally append the default file extension for a database file .DBC. Return this value to the calling method, Init.

Next, test to see if the database name was created and if so, whether or not the specified file exists. If either the database name was not discovered or the file does not exist return a .F. This ensures that the data environment is not instantiated.

Who are my cursors?

If the specified database exists, load the cursors that participate in the data environment. This is accomplished through a call to the LoadCursors method. Notice that the scope resolution operator is not used for this call. The `THIS.LoadCursors()` line of code actually sends execution to the InvItemEnvironment.LoadCursors method where the developer has specified that the v_InventoryItem view and the InvType table should be included in the data environment. When this method finishes executing the aCursors array contains the names of cursor object class definitions that are participants in the data environment.

Create my cursors please !

The next step is to create the cursor objects by FOR looping through each element in the aCursors array and using the ADDOBJECT method of the DataEnvironment class to create those specified

cursor objects. The names of those objects are created by prepending an "o" to the name of the class definition. In this case, the names of the cursor objects are ov_InventoryItem and oInvType.

Creation Of A Cursor Based on the cDynamicViewCursor Class Definition

The first cursor object created is the v_InventoryItem cursor which is based on the cDynamicViewCursor class. At this point, processing flows to the v_InventoryItem class definition. Since no Init code is specified here, processing flows to the cDynamicViewCursor Init method where the following things take place.

Am I abstract?

The first thing tested is if the Dynamic View Cursor object being created is abstract. Since this object is based on the v_InventoryItem class definition, the answer to this question is no and creation of the cursor continues.

Should I use local or remote data?

The next step is to determine whether or not this data environment is using local or remote data. The answer to this question is determined by asking the cursor's parent, which is the InvItemEnvironment class definition what value is in its lUseLocalData property. Remember that the answer to this question was determined when the GetDefaultDatabase method was executed. This is why the cursors must be created after that method is executed. The only way a cursor knows to use local or remote data is to ask its parent. The only way its parent knows is to ask the application. The only way the application knows is when the user, on the User Preference Form, checks the box Use Local Data.

What's my alias?

The next step is to determine what alias to use when addressing this cursor. Notice that the value in the Alias property takes precedence over the value in the cCursorSource property (the value in the cCursorSource propety is provided by the developer in the v_InventoryItem class definition in the ADATAENV.PRG). By default, there is no value contained in the Alias property and the alias of this cursor defaults to the name of the class definition, v_InventoryItem. This is why most of the .ControlSource properties in the controls on a business object have the root name of the view contained in them.

Note that it is easy for the developer to specify another alias by populating the Alias property with the desired value when the cursor's class definition is created. The following code illustrates how this cursor's alias could be overridden to be "InventoryItem".

```
DEFINE CLASS v_InventoryItem AS CDynamicViewCursor
   Alias = "InventoryItem"
   cCursorSource = 'v_InventoryItem'
ENDDEFINE
```

The problem is that many developers do not know the properties available to the cDynamicViewCursor class since this is a programmatic environment and therefore do not know the possibilities available to them.

The power to create different aliases is helpful when the need arises to open the same data source in the same environment for different reasons. To do this, create several class definitions (each with different class names) for that cursor, specify the same CursorSource but provide different Alias names for each. This is normally more useful for tables than views but since a cursor is a cursor the alias property can be used for views as well. This little IF statement makes this possible.

Now ... which view?

Finally, in the next bit of processing, the way Codebook derives the true name of the view is revealed. The contents of the lUseLocalData is interpreted and a .T. is translated into an "l" and a .F. is translated to an "r" and prefixed to the root view name. This is where v_InventoryItem becomes "lv_InventoryItem" or "rv_InventoryItem" depending upon the user's preferences. This value is then placed into the CursorSource property. Notice, that by default, when this occurs the Alias property contains either "lv_InventoryItem" or "rv_InventoryItem".

Give me back my alias !

The next bit of processing takes care of this problem by replacing the Alias property with the desired value for the alias that was stored BEFORE the CursorSource property was populated with the real view name. Now, what was possibly either "lv_InventoryItem" or "rv_InventoryItem" (depending upon the whim of the user) becomes v_InventoryItem for both (or whatever you specified in the Alias property in the class definition). This allows the ControlSource property on the business objects to remain unchanged no matter if local or remote views are being used. This also illustrates that if you override the alias property at this level with your own value, the values in the ControlSource properties for the controls on the business object will have to reflect that value as well.

Who's my database?

Processing then flows to the cCursor Init method where the name of the default database is extracted from the parent data environment object (which got it from the LBSEnvironment property cDefaultDatabase, "LBS" which got it from the QSTART.APP program). Notice that there is no explicit validation whatsoever here. The way a cursor determines to which database it belongs comes straight from you when you created the application using QSTART.APP. The quick start program automatically defined the LBSEnvironment as having a default database name of "LBS". There is no explicit validating that takes place to *ensure* that view really belongs to that database (although VFP may do an implicit check and error out if it doesn't really exist in the specified database).

Dynamic View Cursor Created !

This concludes the creation of a cursor that is based on a local or remove view. Processing now returns to the cDataEnvironment Init method where the next cursor object is created. The reason this cursor can be called a dynamic view cursor is now much clearer. Based upon the value present in the lUseLocalData property the view created as an object and added to the data environment is dynamic, either remote or local in nature, and can be specified by the user, swweeeet!

Creation Of A Cursor Based On The cTableCursor Class Definition

The next step is to create the second cursor that is defined in the data environment and based on the cTableCursor class. When the ADDOBJECT method is executed with the InvType class definition, processing flows to the InvType class definition. Since there is no Init code defined there, processing then flows to the cTableCursor Init method.

Am I abstract ?

Here, the first thing checked is to make sure the object being created is not abstract. Since the object being created is based on the InvType class definition, it is not abstract and processing to create this table cursor continues.

What's my alias ?

The next step in creating the cursor for this table is to explicitly populate the Alias property with the name of the CursorSource. The original documentation says that this is necessary in order to ensure that any relation objects subsequently added to the data environment will be properly setup.

Who's my database?

The final step is to call the cCursor Init method which gets the name of the default database and places it into the Database property of the cursor object. At this point, the last cursor object is created and processing returns to the cDynamicViewCursor Init method.

Notice that this class definition took advantage of the ability to override the Order property of the Cursor object. You can use as many of these properties that you desire, but only if you know they exist.

Continuing With Data Environment Creation ... initially selected alias

The next step in the creation of this data environment is to determine which cursor just created becomes the initially selected alias. This property defaults to 1 but can be overridden by the developer to be any value desired. This numeric value corresponds to the index of the cursor in the aCursors property that is targeted for initial selection. If I had desired the InvType cursor to be the initially selected alias I could have overridden the nIntitalSelectedAlias property in the InvItemEnvironment class definition and given it a value of 2. The setting is incredibly important since the specified alias becomes the one on which all of the business object's create, read, update and deletion functions work upon. It also determines which database is set.

Who am I related to?

The next step is to load the relations in the same fashion as the cursors were loaded. However, since there are no relations defined in the InvItemEnvironment class definition none are created.

Create my cursors and relations

Finally, if the developer wants the cursor/relation objects opened by default (the AutoOpenTables property is set to .T.) then the OpenTables method of the Cursor object is called. It is now that those cursors (tables/views) and relations actually appear in the data environment. Notice that the call to open tables did not use the scope resolution operator. This means that you can override the behavior of the OpenTables method by programmatically adding these methods at any class definition in the class hierarchy. This provides you with the ability to perform pre- and post-processing on the tables during the opening process.

Conclusion

The data environments, once their behavior is dissected in detail, are really not that bad and illustrate that Codebook provides the ability to do just about anything desired with tables, views and relations since the complete functionality of the DataEnvironment, Cursor and Relation FoxPro base classes is available to the developer, but only if they are familiar with the power lying therein..

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: cblanke@norfolk.infi.net, www.savvysolutions.com, CompuServe 76132,2575

Fix for BuildMetaData()

This is a fix for a bug in the BuildMetaData() Function found in SETUP.PRG

Paul Tew

The BuildMetaData function has a bug in the SQL statement that selects records from the Project File. The SQL Statement (found near the end of the function) reads like this:

```
SELECT name, type ;
       FROM _project ;
       WHERE !DELETED() ;
       AND type = "V" ;
       OR type = "P" ;
       ORDER BY type ;
       INTO CURSOR cTemp
```

This results in records deleted from the project file of type “P”, being included in the list of files which subsequently has a SET PROCEDURE TO <filename> applied. In other words, we try to SET PROCEDURE TO a filename which may not exist, and we certainly don’t want included as a procedure file.

This code should have the OR statement bracketed, or perhaps use the INLIST function as follows.

```
SELECT name, type ;
       FROM _project ;
       WHERE !DELETED() ;
       AND INLIST(type, "V","P") ;
       ORDER BY type ;
       INTO CURSOR cTemp
```

Paul Tew can be reached via Tel +44 (161) 283 3119 , 101533.1767@compuserve.com, p2.panacea@zen.co.uk.

For You Potential Authors

The Codebook News will only survive if the readers themselves participate. This is a template for all of you potential authors to use when submitting articles for publication.

Charles T. Blankenship

This is regular text. The first paragraph does not have an indented first line but every other paragraph in the article should be indented. The only exceptions: after chunks of code, subheads, and tables.

This is more regular text - note the indented first line - done with a simple tab, of course. Remember that every style in this style sheet starts off with an underscore. Thus, the style for this paragraph is _body text.

This is more regular text blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah

This is a subhead - it uses style of _subhead 1 and doesn't have any caps

This is more regular text blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah

```
This is code.
It uses style of _code.
This is more code
One rule about code, do NOT use tabs - use spaces - to indent the code
The reason this style sheet was created was to "give people more room" and prevent
lines of code from being restricted to 50 characters.
```

This is more regular text blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

Sometimes you will want to list a number of interesting things in a list. We call them bullets. Here's how we do this - four bullets:

- This is bullet 1.
- This is bullet 2.
- This is Steve McQueen's Bullet.
- As you can see, they have style of _body text but we use the Bullet button on the Word toolbar.
-

The next interesting item is - yes, a VISUAL! In other words, a picture! And remember, a picture is worth a thousand words (even though your checks don't reflect it). Please endeavor to use a picture or two in your article. Also remember to reference the figure in your text in bold, like this (see Figure 1). Here's how we indicate we have a picture:

05RALPH1.BMP -

Figure 1. "The user can choose between five options in the Widget dropdown listbox. This caption uses the _figure caption style."

The next chunk of text is a table. Note that we use tabs, not Word's Table function. Insert tabs for each column. Don't rely on the default tabs. There should be only one tab between each column.

Table	Table	Table
Table	Table	Table

This is more regular text blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah

This is more regular text blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah

This is a closing paragraph. It's a good idea to sort of sum up what you said, and perhaps give the reader a lead toward what their next step would be...

This is the bio for the author. You can say anything you want. By the way, when you include contact info, note the following format - list phone numbers and email addresses, including the period in a CIS address instead of the comma. 757-853-4465, fax757-853-4465,76132,2575@compuserve.com, cblanke@norfolk.infi.net.