Clarion Magazine

For marketing your Applications
and Developer Accessories or to
purchase other 3rd Party Tools . . .
Developer PLUS tm

Ads · Comments · Writers! · Privacy · Contacts · PDFs · Freebies · Open Source

PDF for November, 2002

All Clarion Magazine articles for November, 2002 in PDF format.

Posted Monday, December 02, 2002

Data Structures and Algorithms Part XII - Trie This

In this article Alison Neal introduces a data structure known as the Trie, which is basically a type of general tree, containing words rather than numbers. The Trie is an immensely useful data structure when storing strings in memory. The Trie has been used for such things as computerized Boggle and Yahtzee games, and file compression algorithms.

Posted Wednesday, December 04, 2002

Parsing Strings In ASCII Files

Konrad Byers' recent article on a class for accessing and processing ASCII files really spoke to Steve Parker. In this article, Dr. Parker combines that class with some string parsing code to extract field data from an ASCII file record.

Posted Thursday, December 05, 2002

Viewing An Excel Spreadsheet In A Clarion Browse

You can get data for a browse box from some surprising sources. Ayo Ogundahunsi shows how easy it is to use a linked server to display data from an Excel spreadsheet in a Clarion browse.

Posted Thursday, December 05, 2002

Weekly PDF For December 1-7, 2002

All ClarionMag articles for December 1-7, 2002 in PDF format.

Posted Monday, December 09, 2002

Countdown To CLARION 6 Early Access Release

An Early Access (EA) release of Clarion 6 is expected during the week of December 16th. And yes, it will be Clarion 6 not 5.6, as this release was deemed to have too many features for a dot release. Bookmark this page for the latest news on C6EA.

Posted Monday, December 09, 2002

SUBSCRIBE

6 MONTHS $45

1 YEAR $80

2 YEARS $150

Subscribe Now

News

SealSoft xAnalogClock 1.2

Clarion 6 EA Program Now Full

Another RADrace Victory For Clarion!

File Manager 3 Beta 9a

New ImageEx 2 Demo

ConVic 2003

CPCS Christmas Schedule

SealSoft New Year Discount

wPDFControl Wrapper

New Icetips Bulletin Board

Clarion Template/API Forum

Clarion Source Code For Sale

Clarionfoundry Open To Public Again

IceTips December Newsletter

Icetips Holiday Specials

Possible Florida UG Conference

ImageEx2 Beta 3

Data Structures and Algorithms Part XIII - Trie Hard

In her last installment, Alison Neal introduced a data structure called the Trie, which is used for storing strings in computer memory. In this article Alison continues her discussion of the Trie, and covers some of the other methods that are contained in the Trie Class, namely the Search, Print and Kill methods.

Posted Thursday, December 19, 2002

DNA for Clarion: Manipulating Browse Cells With A VLBPROC (Part 1)

Virtual List Boxes (VLBs) are one of the least-understood and most under-appreciated features of the Clarion language. In this two-parter, Stephen Bottomley explains VLBs, and introduces a class that you can use standalone, or to control the display of an existing browse. Part 1 of 2.

Posted Thursday, December 19, 2002

DNA for Clarion: Manipulating Browse Cells With A VLBPROC (Part 2)

Virtual List Boxes (VLBs) are one of the least-understood and most under-appreciated features of the Clarion language. In this two-parter, Stephen Bottomley explains VLBs, and introduces a class that you can use standalone, or to control the display of an existing browse. Part 2 of 2.

Posted Friday, December 20, 2002

The Clarion Advisor: Displaying Clarion Dates In Excel

If you have a CSV file or other data that contains an unformatted Clarion date, and you wish to view the date in Excel, you'll need to convert the Clarion standard date to an Excel date. Here's how you do it.

Posted Friday, December 20, 2002

Web Validation From Your Clarion App Using NetTalk

Recently, Mark Riffey had a need to for one of my Clarion programs to access a SQL database, hosted on the web, in order to determine if the customer's access to a service had expired. It was a fairly simple task using CapeSoft NetTalk, as Mark demonstrates.

Posted Friday, December 20, 2002

CLASSy ASCII File Importing

Earlier this month Steve Parker wrote an article on importing ASCII files into a database using Konrad Byers' ASCII file class. In this article, Steve describes a class by Dave Harms that makes fixed record length ASCII importing configurable at runtime.

Posted Friday, December 20, 2002

S.C.A. Micro Legacy Templates

ClarionPost Reopens

Last Chance For List & Label Discount

Simsoft Christmas Stocking - Save $58US

International Clarion Meetup Day

Save 10% On Image Man OCX

RInstall V1.f (Beta) Update

INN Bio & News for 3-Dec-2002

Clarion Handy Tools Newsgroup Server

Shapemaker SMX

Subject: Nextage Imaging Update

xFText v2.0 Released

DOS Printer v7.4

BigTamer Update

ZipApp Free Backup Program

Search the news archive

ClarionMag Office Holiday Schedule

The Clarion Magazine office is now closed for the holidays, and will reopen January 6, 2003. Subscriptions and renewals will be processed automatically, as usual. I will respond to emails and phone messages as soon as possible when the office opens. A Merry Christmas and Happy New Year to all!
Dave Harms, Publisher

Posted Saturday, December 21, 2002

Looking for more? Check out the **site index**, or **search the back issues**. This site now contains more than 700 articles and a total of over a million words of Clarion-related information.

Clarion Magazine

Data Structures and Algorithms Part XII - Trie This

by Alison Neal

Published 2002-12-04

In this article I will introduce a data structure known as the *Trie*. A Trie is basically a type of general tree, containing words rather than numbers. The Trie is an immensely useful data structure when storing strings in memory. The Trie has been used for such things as computerized Boggle and Yahtzee games, and file compression algorithms.

The best possible illustration for the Trie's use, though, is in Internet search engines. The Trie is built to contain whole words and maintain a count of how many times a word occurs. In so doing this structure makes the search of web pages and the weighting of pages by the recurrence of a particular word quite simple.

It’s easy to imagine an Ordered Linked List being used for this purpose as well; all I’d have to do is store a word in each node and add a count variable to the Ordered List structure, so that I could count the number of word recurrences. However, that would be a very slow way of going about things in comparison to using the Trie structure.

In my previous articles on Trees I pointed out that when a search is performed on an Ordered Linked List, every node has to be visited (starting at the head) until the logical position of the item being searched for is reached. I then showed that a Tree could make this search far more efficient as, if the value being sought was lower than the current node, the search would move left, or if the required value was higher than the current node, then the search would move to the right. This could effectively halve the search time of an Ordered Linked List.

The Trie takes this idea one step further. It is a given that in the English language there are only 26 letters (A to Z). So the value of the first letter in a word is only ever going to be between 1 and 26, and the value of the second letter is also going to be between 1 and 26, the same for the third, and so on. This naturally lends itself to a structure containing an array of 26.

Here is the definition of the Trie Structure:

```
nBranches   EQUATE(27)
maxLen      EQUATE(80)

dNode       CLASS,TYPE
n             ULONG(0)
s             CSTRING(maxLen+1)
            END

bNode       CLASS,TYPE
n             ULONG(0)
p             LONG(0),DIM(nBranches)
            END

root        LONG(0)
currD       &dNode
currB       &bNode
```

The first thing to note about this structure is that there are two types of nodes, bNodes (branch nodes) and dNodes (data nodes), and neither of them contains a reference variable. So how are they being linked? When a new node is created, rather than storing the reference variable, I am storing the address of the new node in the `LONG` variables. This is what allows me to use two different types of node within the same structure.

In C/C++ I would have to cast the pointer types from one type of node to the other; in Clarion I can just read the address and assign it to a reference variable as required. Note that both the branch nodes and the data nodes contain the variable n, which means I can assign a branch node to a data node type reference variable and still check to see whether n has a value. Thus, I’ve used the n to identify whether a node should be a branch node or a data node (see Figure 1.).

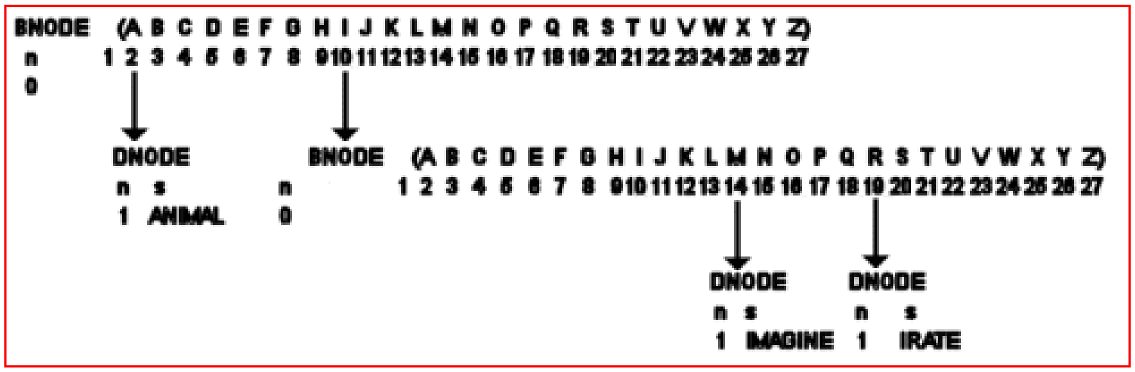


Figure 1. A Trie with three words added.

With data nodes the n variable always contains the count of word occurrences, which will be at least 1, as the node wont exist unless the word in question has occurred at least once. The branch nodes will always be zero because they don't contain any words.

Figure 1 shows how the Trie will look if the words "ANIMAL"," IMAGINE" and "IRATE" are added to it. "ANIMAL" is the only word beginning with the letter A, so it has a data node of its own. "IRATE" and "IMAGINE", however, both start with the letter I, so they require a branch node for the I place keeper, and have a data node associated with their second letter M and R.

So how did I do it?

She scores a Trie

The Trie class I have written (downloadable at the bottom of this article) includes the following methods:

Method	**Description**
`Init`	Open the file to be read and insert words into the Trie
`Kill`	Dispose of the Trie
`Inst`	Insert a word into the Trie
`Search`	Search the Trie for a specific word
`Pr`	Print the contents of the Trie

The `init` method is reasonably simple; it opens a data file and scans each line for legitimate words (it treats all non-alphabetic characters as white space, and converts alphabetic characters to upper case). On finding a word it calls the `Inst` (Insert) method, passing the word to be added, the root of the Trie (a `LONG`, which is zero for the first word and after that the value returned by `Inst`) and a position variable to mark the character in the string that is of concern in this recurse, since `Inst` will be called recursively. The first time `Inst` is called for any word, of course, the position should be 1.

```
Trie.inst                PROCEDURE(*STRING s,LONG r,*ULONG m)
PP  LONG(0)
q   LONG(0)
t   &DNode
p   &BNode
j   LONG(0)
i   LONG(0)
    CODE
    pp = r
    q = pp
    IF q = 0 THEN RETURN SELF.NewNode(s).

    t &= (q)
    IF t.n > 0
      IF s = t.s
        t.n += 1
        RETURN q
      END
      pp = SELF.NewNode(' ')
      q = pp
      j = SELF.pos(t.s[m])
      p &= (q)
      p.p[j] = ADDRESS(t)
    END
    i = SELF.pos(s[m])
    m+=1
    p &= (q)
    p.p[i] = SELF.inst(s,p.p[i],m)
    q = pp
    RETURN q
!---------------------------------------------------------
Trie.NewNode               PROCEDURE(STRING s)
!---------------------------------------------------------
    CODE
    IF s
      SELF.CurrD &= NEW(dNode)
?     ASSERT(~SELF.CurrD &= NULL)
      SELF.CurrD.n = 1
      SELF.CurrD.s = s
      RETURN ADDRESS(SELF.CurrD)
   ELSE
      SELF.CurrB &= NEW(bNode)
?     ASSERT(~SELF.CurrB &= NULL)
      SELF.CurrB.n = 0
      RETURN ADDRESS(SELF.CurrB)
   END
   RETURN 0

!---------------------------------------------------------
Trie.pos               PROCEDURE(STRING ch)
!---------------------------------------------------------

    CODE
    IF ch
      RETURN val(ch[1]) - val('A') + 2
    END
    RETURN 1
```

Here's what's happening in the example shown in Figure 1. When I first add the word "ANIMAL", the root is going to be equal to 0, as nothing has been added before, so the `NewNode` method is called passing the word "ANIMAL" and returning the address of the data node that is created. The Trie now looks like Figure 2.

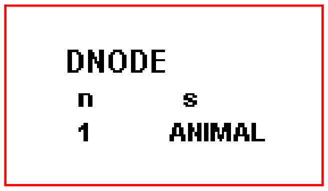


Figure 2. The Trie with one word added

As there is only one word there is no requirement for a branch node to start with.

On the next word insertion, `s` is "IMAGINE", `m` is 1, and the root contains the address of the node shown in Figure 2. This time `q` is not equal to zero as it now contains a memory address. Now `t` is made to refer to the root node and a check is made to see whether the node is a data node, or a branch node. Remember the branch node should always have an n value of 0, and a data node will be at least 1.

`T` is a data node, but the string it holds does not match the string that I want to add to the Trie – it is not a recurrence of the same word. If it were a recurrence then `n` would have been incremented by one, and the method would have returned.

A new node is now created, and as a string is not passed the `NewNode` method creates a branch node, which contains an array of `LONG`s. `Q` is then made to equal this new branch node, as `Q` is now the new root node, the address of the old root node (see Figure 2) needs to be stored in the appropriate position. The `pos` method provides this position in the array. Figure 3 shows how the Trie looks at this stage.

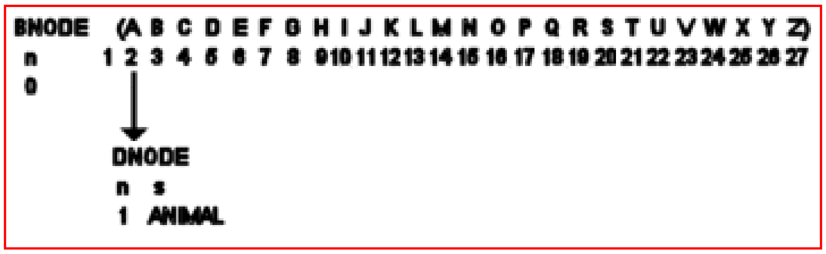

BNODE (A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)
n 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
0
DNODE
n s
1 ANIMAL

Figure 3. The Trie with two words added

The `init` method hasn't finished yet, as it still has to add the word "IMAGINE" to the structure. The appropriate position in the structure based on the first letter in the word "I" is provided by a call to the `pos` method (10); then a recursive call is made to the `inst` method passing "IMAGINE", the appropriate node in the structure (position 10 in the array), and `m` which is now equal to 2.

In this call to the inst method, r is now 0, because position 10 in the array has had nothing assigned to it yet, so a new data node is created with the word "IMAGINE" assigned to it, and on returning to the original inst method, the node's address is assigned to position 10 in the array, giving Figure 4.

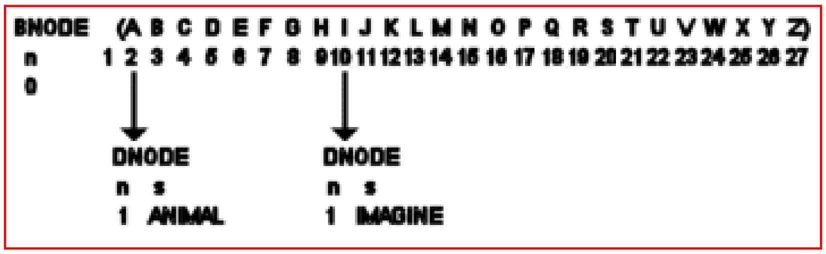


Figure 4. The Trie with two words added

Now I want to add the word "IRATE" to the structure. The inst method is called passing "IRATE", the address of the root branch node, and `m` the string position holder (1).

This time through `q` does not equal 0, and the root node's `n` variable is not greater than 0 - as with branch nodes, `n` is always equal to 0. The appropriate position of the first character in the word "IRATE" (`I = 10`) is found, `m` is incremented to 2 (to signify that the next call will care about the second letter of the word) and a recursive call is made to the inst method, passing the Address of the tenth node in the array (the node containing the word "IMAGINE").

Again `q` is not equal to 0, but the n variable is greater than 0. The word to be added and the word already stored do not match and so n is not incremented. Instead a call is made to the NewNode method without passing a string, so that the address of a new branch node is returned.

This new branch node will fill the position currently held by the data node containing the word "IMAGINE," in the same way that the last branch node took the root position when the word "IMAGINE" was first added. Remember how the data node containing the word "ANIMAL" was then linked via address to the appropriate branch? This time the data node containing the word "IMAGINE" is linked, and a new data node is added in a recursive call for the value "IRATE". This finally gives the completed Trie, illustrated in Figure 4.

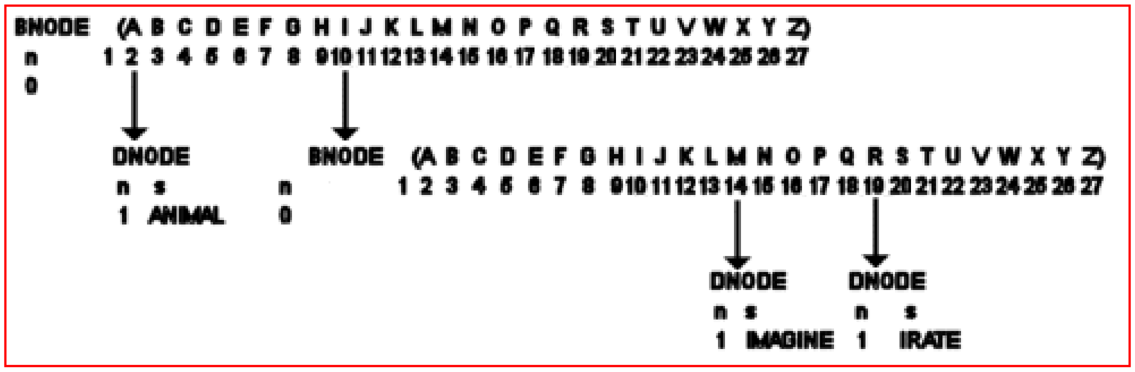


Figure 4. The Trie with all three words added.

Summary

The Trie data structure is on my list of fun things to do with words. It may seem overwhelming at first with the two different types of node being linked together by address, but once it is understood it is simplicity itself. The Trie, as I've mentioned, is extremely useful for ordering words, counting recurrences of words in text, and for games such as boggle and Yahtzee.

In my next article I will cover the other Trie methods, including `search`, `print` and `kill`.

Download the source

Alison Neal has been using Clarion since 2000, whilst working for Asset Information Systems (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarioNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, Manufacturers, Military & Government, Legal & Financial, and Retail.

Reader Comments

Add a comment

Clarion Magazine

Parsing Strings In ASCII Files

by Steven Parker

Published 2002-12-05

Konrad Byers' recent article on a class for accessing and processing ASCII files *really* spoke to me. I am constantly dealing with all manner of ASCII files, both fixed length (which his class deals with) and delimited, in more varieties than Carter has little pills.

I support an application which uses a variety of ASCII files to import employee data into a customer file (different payroll/personnel systems, different ASCII files). And for each import, I have to declare a file layout and a `NAME()` variable. I have to make sure the variable is declared in the right way and in the right place or I'll get compiler or runtime errors. I have to make sure the variable is initialized at the correct time. I have to check to see whether the file is actually there (the import is run automatically as part of an end of day procedure so user selection is not an option). "Etc., etc. and so forth."

All of this, just as Konrad observes in his article, is a royal pain in the keyboard.

Konrad's class eliminates almost all of the work I need to do when reading and processing these files. I just instantiate the class (create an object), create a variable (to contain a line of text, i.e. the record read) and I'm off to the races.

I don't have to worry about `NAME()`. I don't have to do much error checking either; the class handles it.

Following his example, all I have to do is put

```
Include('AnyAscii.inc'), ONCE
```

in the Global Data area of my application and

```
InputFile    AnyAsciiFileClass   !instantiate
AsciiText    CString(257)        !"record"
```

in the data section of my procedure. Then, my processing code is literally straight out of the article (okay, slightly modified for my coding style):

```
If InputFile.OpenFile('\incoming\data\giftshop.out')
  Return !there is an error, don't process
End
Loop While ~InputFile.Read(AsciiText)
  !make field assignments here
End
InputFile.CloseFile()
```

Konrad’s class covers it all. Except for one minor detail.

The file I am importing (to use a real example) looks like this:

```
IMPASCII    File,Pre(IMP),Driver('ASCII''),Name(Pattern_)
Record       Record
EmployeeID     String(16)
BadgeNumber    String(16)
LastName       String(15)
Firstname      String(15)
NotUsed        String(15)
Balance        String(10)
             End
            End
```

But the `AnyAsciiFileClass` only knows about a single string variable. I can’t very well make field assignments to `CUS:Number`, `CUS:BadgeNumber`, `CUS:LastName`, etc., from a single 256 character field.

What I need to somehow do is to divide up the `AsciiText` variable. I need to divide it into readily identifiable chunks, chunks that correspond to my file fields.

What I need to do is parse `AsciiText` into useable pieces.

Clarion provides three ways to parse strings: The `SUB` function, string slicing, and `OVER` declarations. Let’s look at them.

Using SUB

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

Supported since there was a Clarion, "The **SUB** procedure parses out a sub-string from a *string* by returning *length* characters from the string, starting at *position*." `SUB` requires I know where

I want to start reading (*position*) and how many characters I want to read (*length*). For a batch import, this is fairly straightforward.

For the file structure above, my processing code would be:

```
CUS:Number    = SUB(AsciiText,1,16)
CUS:Badge     = SUB(AsciiText,17,16)
CUS:LastName  = SUB(AsciiText,33,15)
CUS:FirstName = SUB(AsciiText(48,15)
CUS:Balance   = SUB(AsciiText,78,10)
```

And if I count wrong? I laid this out on a coding sheet, the kind with numbered columns, and still got it wrong twice.

`SUB` allows me to start from either end of the string. That means

```
CUS:Balance = SUB(AsciiText,-1,10)
```

is exactly the same, in this case, as

```
CUS:Balance = SUB(AsciiText,78,10)
```

and

```
SUB(AsciiText,-24,15) = SUB(AsciiText(48,15)
```

The docs tell me that `SUB` uses more memory (and, therefore, is slower) than string slicing:

String slicing (a.k.a. implicit arrays)

Starting sometime in the CDD3 cycle, all string variable types (Strings, CStrings and PStrings) acquired an implicit array declaration. This means that

```
MyString    String(20)
```

is identical to

```
MyString    String(1),DIM(20)
```

That is, a string can also be addressed as if it were an array. In practice, this means that I can read the first character of `MyString` with

```
MyString[1]
```

which is the same as `SUB(MyString,1,1)` or the fifth through eighth with

```
MyString[5 : 8]
```

which is identical to

(`SUB(MyString,5,3)`)

In the case of the file I am importing, I know that `EmployeeID` is the first field, therefore, it begins at position 1. I know that it is 16 characters long. So I can get the customer number this way:

```
CUS:Number = AsciiText[1 : 16]
```

I know that `BadgeNumber` begins immediately after `EmployeeID`, so it begins at position 17. It is 16 characters long, so it must end at position 32 (count 16 starting with 17). So, this is the slice:

```
CUS:Badge = AsciiText[17 : 32]
```

And so forth. Note that I leave a space on either side of the colon. This is because constants, variables and expressions may all be used in an implicit array (`SUB` also supports variables and expressions). For example, in the following code, `X` contains everything from the beginning of the string up to but not including the first pipe character:

```
X = MyString[1 : Instring('|',MyString,1,1) - 1]
```

If a variable or expression is used, you must separate it from the colon with a space, or the compiler may interpret the colon as part of a label. Because of the possibility of an error, I just adopted the habit of always including the space whenever I slice strings.

The big drawback to string slicing (and this *is* documented) is that string slicing does no bounds checking. If one of the implicit array elements in an expression is out of range, very weird things are going to happen.

"Very weird" meaning you will *not* get a compiler error or warning. But you may see fairly bizarre (and almost impossible to debug) behavior. (Yeah, I got the T shirt.)

The `SUB` function and string slicing do the same job and give the same results. But string slicing is often more readable.

A really cool use for string slicing

When I look at the bar code on my pastrami wrapper, I see 200202 004632. But the last time I bought pastrami, it was 200202 008674.

Two UPC codes for the same product? I don't think so.

The Uniform Commercial Code (UCC) includes a standard for random (or variable) weight codes (see Guideline 11: Variable Measure (Includes Random Weight and Count)).

Under this standard, if the first character of a bar code is "2," the bar code is a random weight bar code. The third through sixth characters are the "Price Lookup Unit" (i.e., an internal inventory identifier assigned by the seller) and eight through 11 are the price.

If I were writing a POS application where barcode labels were produced by scales, I'd code my price calculations this way:

```
IF EntryString[1] = '2'       !random weight bar code
  INV:PLU = EntryString[3:6]  !prime lookup
  Get(ABINVTRY,INV:PLUKey)    !lookup from inventory
  If ~ErrorCode()             !calculate quantity purchased
    EnteredQuantity = (EntryString[8:11]/100) / INV:UnitPrice
  Else
    Message('Item not found in inventory', |
              'Error',ICON:Hand)
    Return
  End
Else
  !standard inventory validation here
End
```

OVER (set shared memory location)

`SUB` is an old friend and most languages have a similar function. String slicing is (relatively) new. But, `OVER`ing a variable probably is not one of the things you think of when you think of parsing strings.

Until Kurt Pawlikowski demonstrated it to me, I was among those who did not think of `OVER` in connection with string parsing. Of course, I have the additional "advantage" of having a long time mental block about this attribute.

`OVER`, as I vaguely recall, used to be described as a way of allowing two variables to use the same memory. The current description in the Language Reference Manual is ***much*** more helpful: "Allows one memory address to be referenced two different ways."

This, I understand. It is just like a `Group` structure, which allows you to refer to individual fields, or all fields at once: multiple ways of referring to fields: "A `GROUP` structure allows multiple variable declarations to be referenced by a single label."

Consider the example given in the docs:

```
CustNote     FILE,PRE(Csn)    !Declare CustNote file
Notes          MEMO(2000)    !The memo field
Record         RECORD
CustID           LONG
             . .
CsnMemoRow   STRING(10),DIM(200),OVER(Csn:Notes)
```

`CsnMemoRow` is the same as a set of 200 ten-character blocks laid end –to end. And, it begins at the same place (in memory) that `CSN:Notes` begins. So, `CSN:Notes[1 : 10]` = (is the same as `CsnMemoRow[1]`and `CSN:Notes[21 : 30] = (`is the same as) `CsnMemoRow[3]`

> (Actually, I’m not certain string slicing works on memos. At least in the past, I have had trouble slicing memos. In cases where I needed to parse a memo, I declared a string, of the same size, `OVER` the memo and sliced the string.)

Now, suppose I declare a `GROUP` and I declare it `OVER` the `AsciiText` variable from my original ASCII record example. That means that the fields within the `GROUP` will contain discrete parts of `AsciiText` (since a `GROUP` is, essentially, a string, a sort of super-string). Suppose I create the fields within the group to match the format of the incoming file:

```
Incoming     Group,PRE(INC),OVER(AsciiText)
EmployeeID     String(16)
BadgeNumber    String(16)
LastName       String(15)
Firstname      String(15)
NotUsed        String(15)
Balance        String(10)
             End
```

My processing code, then, becomes:

```
If InputFile.OpenFile('\incoming\data\giftshop.out')
  Return !there is an error, don't process
End
Loop While ~InputFile.Read(AsciiText)
  CUS:Number = INC:EmployeeID
  CUS:Badge  = INC:BadgeNumber
  !Etc.
End
InputFile.CloseFile()
```

And this looks remarkably like my existing code. Very comforting, very. (See the demo app for a real example.)

Recasting the random weight bar code example from above, I could have a local data declaration like this:

```
RanWeight    Group,PRE(LOC),OVER(EntryString)
NotUsed        String(2)
PLU            String(4)
AlsoNotUsed    String(2)
Price          String(4)
WhoCares       String(1)
             End
```

And my processing becomes:

```
IF EntryString[1] = '2'    !random weight bar code
  INV:PLU = LOC:PLU        !prime lookup
  Get(ABINVTRY,INV:PLUKey) !lookup from inventory
  If ~ErrorCode()          !calculate quantity purchased
    EnteredQuantity = (LOC:Price/100) / INV:UnitPrice
  Else
    Message('Item not found in inventory', |
              'Error',ICON:Hand)
    Return
  End
Else
  !standard inventory validation here
End
```

I don’t know how the two approaches compare for efficiency - the tradeoff is the extra data declaration(s) vs. whatever processing string slicing requires.) But the OVER approach certainly is eminently readable.

Summary

For simple parsing needs, SUB and string slicing are fast and efficient. On the other hand, OVER is not the first thing you would think of when parsing strings. But, as Kurt demonstrated to me, when you need to do high speed, repetitive, sequential parsing ("batch processing"), it fills the bill quite nicely, Quite nicely indeed and combined with Konrad’s classes, looks very much like a "method of choice" in the making.

Download the source

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.

Reader Comments

Add a comment

Great article, I string slice memos all the time. It works...

Clarion Magazine

Viewing An Excel Spreadsheet In A Clarion Browse

by Ayo Ogundahunsi

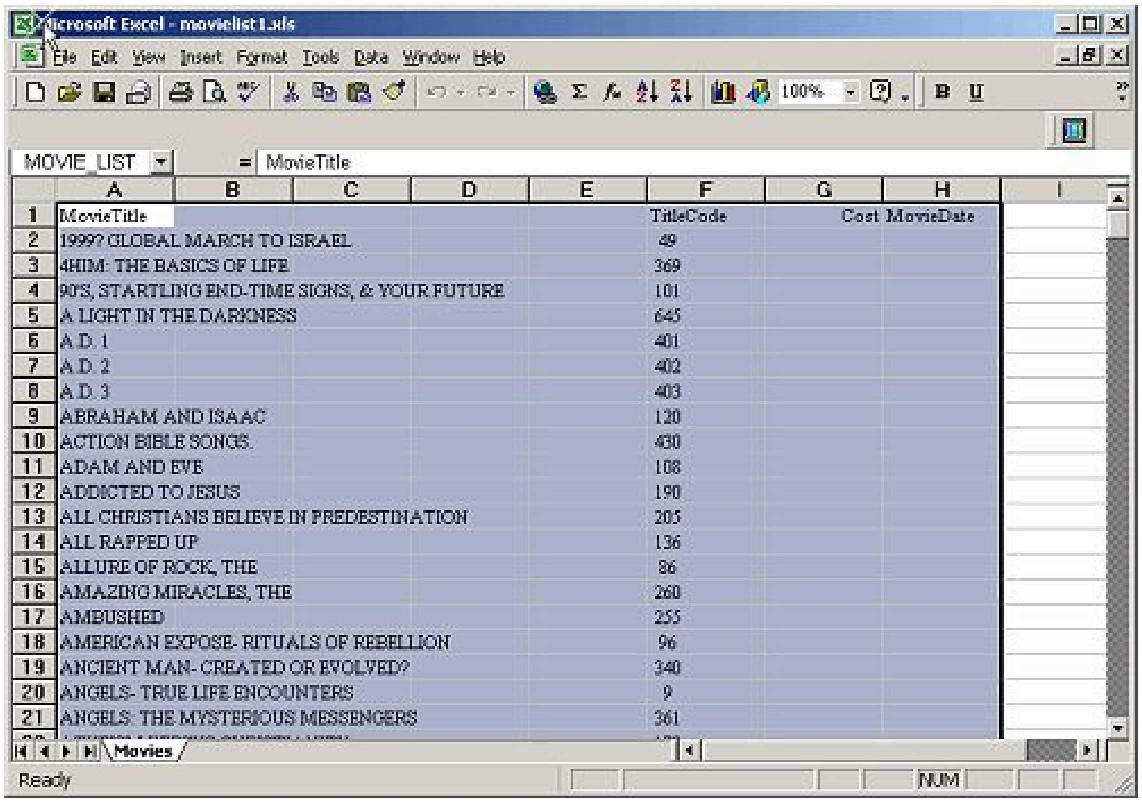
Published 2002-12-05

In my article – Migrating The Inventory Application To SQL Server, Part 4, I discussed Linked Servers, and used a Linked Server to import a list of States from an Excel Spread sheet into SQL Server. I want to do something different in this article. Using a Linked server, I will view the Excel data in a Clarion Browse, *without* first importing the data into SQL server. This is very simple, as you will soon find out.

There are five steps required to view an Excel spreadsheet in a Clarion browse: Set up the spreadsheet; set up the Linked Server; create a view; import the view into Clarion; create the Clarion browse.

Step 1: Setting up the Spreadsheet

In order to be able to access the cells in the spreadsheet, I need to define a range for the data I need to view. In the sample Excel file, I have assigned the label `MOVIE_LIST` to the cell range `A1:H364` (see Figure 1).


Microsoft Excel - movielist1.xls
File Edit View Insert Format Tools Data Window Help
100%
B U
Movies
Ready
NUM

MOVIE_LIST = MovieTitle

	A	B	C	D	E	F	G	H
1	MovieTitle					TitleCode	Cost	MovieDate
2	1999? GLOBAL MARCH TO ISRAEL					49		
3	4HIM: THE BASICS OF LIFE					369		
4	90'S, STARTLING END-TIME SIGNS, & YOUR FUTURE					101		
5	A LIGHT IN THE DARKNESS					645		
6	A.D. 1					401		
7	A.D. 2					402		
8	A.D. 3					403		
9	ABRAHAM AND ISAAC					120		
10	ACTION BIBLE SONGS.					430		
11	ADAM AND EVE					108		
12	ADDICTED TO JESUS					190		
13	ALL CHRISTIANS BELIEVE IN PREDESTINATION					205		
14	ALL RAPPED UP					136		
15	ALLURE OF ROCK, THE					86		
16	AMAZING MIRACLES, THE					260		
17	AMBUSHED					255		
18	AMERICAN EXPOSE- RITUALS OF REBELLION					96		
19	ANCIENT MAN- CREATED OR EVOLVED?					340		
20	ANGELS- TRUE LIFE ENCOUNTERS					9		
21	ANGELS: THE MYSTERIOUS MESSENGERS					361		

Figure 1. Setting Range in Excel Spreadsheet

Step 2: Setting up the Linked Server

I won't go into the detail of to seting up a Linked Server as I've done so in an earlier article. The MS SQL script below will create a Linked Server to an Excel Spreadsheet located in: `C:\INVNTORY\MovieList1.xls`.

```
sp_addlinkedserver N'MovieLExcel', N'Jet 4.0',
     N'Microsoft.Jet.OLEDB.4.0',
     N'C:\INVNTORY\MovieList1.xls', NULL, N'Excel 8.0'
GO
sp_addlinkedsrvlogin N'MovieLExcel', false, sa, N'ADMIN', NULL
GO
```

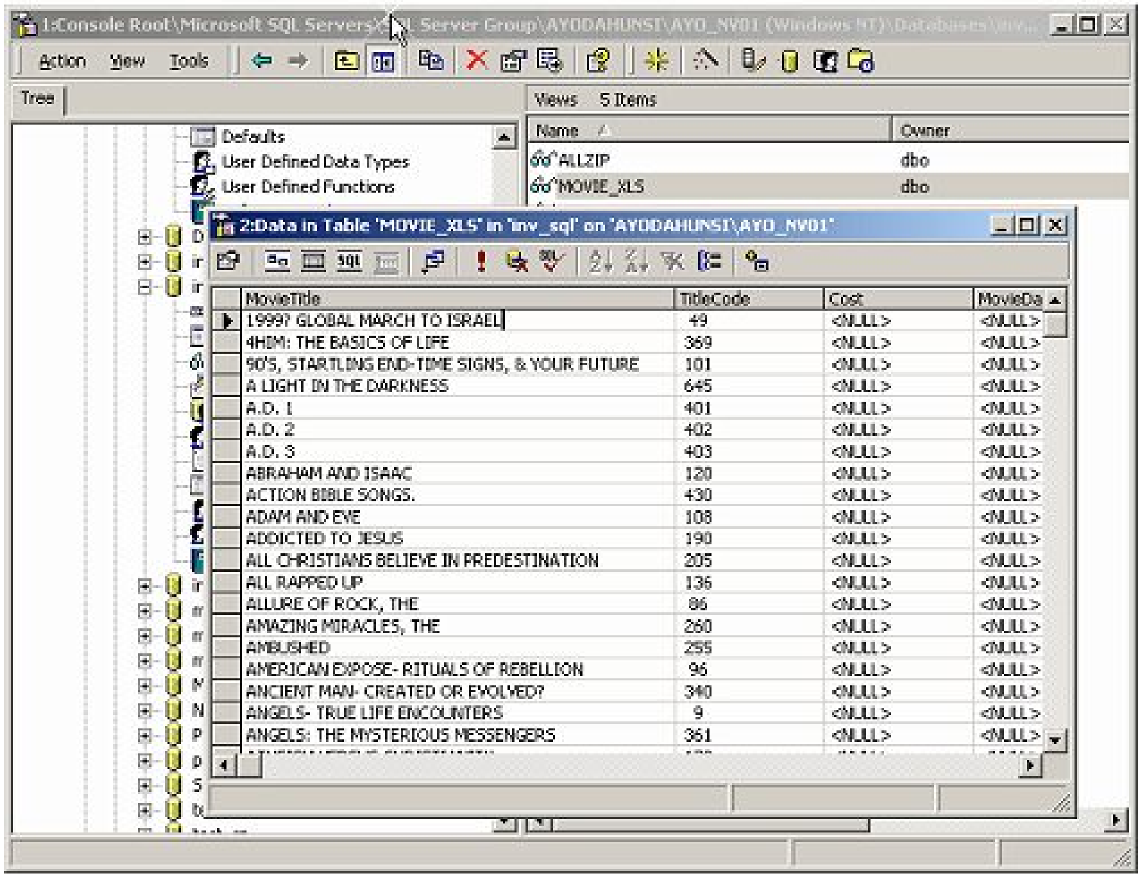
Copy this code into your Query Analyzer and execute it.

Step 3: Create a view

The next thing I need to do is create a SQL View. Copy, paste, and execute the script below to create a View.

```
CREATE VIEW MOVIE_XLS as
SELECT MovieTitle, TitleCode,Cost,MovieDate
FROM   MovieLExcel...MOVIE_LIST MOVIE_LIST1
```

The View I created looks like Figure 2 when queried from the Enterprise Manager.


1:Console Root\Microsoft SQL Servers\SQL Server Group\AYODAHUNSI\AYO_NV01 (Windows NT)\Databases\inv...
Action
View
Tools
Tree
Defaults
User Defined Data Types
User Defined Functions
Views 5 Items
Name
Owner
ALLZIP
dbo
MOVIE_XLS
dbo
2:Data in Table 'MOVIE_XLS' in 'inv_sql' on 'AYODAHUNSI\AYO_NV01'

MovieTitle	TitleCode	Cost	MovieDa
1999? GLOBAL MARCH TO ISRAEL	49	<NULL>	<NULL>
4HIM: THE BASICS OF LIFE	369	<NULL>	<NULL>
90'S, STARTLING END-TIME SIGNS, & YOUR FUTURE	101	<NULL>	<NULL>
A LIGHT IN THE DARKNESS	645	<NULL>	<NULL>
A.D. 1	401	<NULL>	<NULL>
A.D. 2	402	<NULL>	<NULL>
A.D. 3	403	<NULL>	<NULL>
ABRAHAM AND ISAAC	120	<NULL>	<NULL>
ACTION BIBLE SONGS.	430	<NULL>	<NULL>
ADAM AND EVE	108	<NULL>	<NULL>
ADDICTED TO JESUS	190	<NULL>	<NULL>
ALL CHRISTIANS BELIEVE IN PREDESTINATION	205	<NULL>	<NULL>
ALL RAPPED UP	136	<NULL>	<NULL>
ALLURE OF ROCK, THE	86	<NULL>	<NULL>
AMAZING MIRACLES, THE	260	<NULL>	<NULL>
AMBUSHED	255	<NULL>	<NULL>
AMERICAN EXPOSE- RITUALS OF REBELLION	96	<NULL>	<NULL>
ANCIENT MAN- CREATED OR EVOLVED?	340	<NULL>	<NULL>
ANGELS- TRUE LIFE ENCOUNTERS	9	<NULL>	<NULL>
ANGELS: THE MYSTERIOUS MESSENGERS	361	<NULL>	<NULL>

Figure 2. Querying created View

Step 4: Import the view into Clarion

Now that you can access the spreadsheet from a view, it is quite easy to put the spreadsheet data into a browse. First, using the Clarion Synchronizer (or File|Import) you import the view into the Clarion Dictionary.

While experimenting, I discovered that all fields are imported into Clarion as `CSTRING(256)`. So you can create the table manually if you don't want to mess around with the synchronizer. Just remember to use an external name that matches the name of the View created in Step 3.

Here is what my file structure looks like:

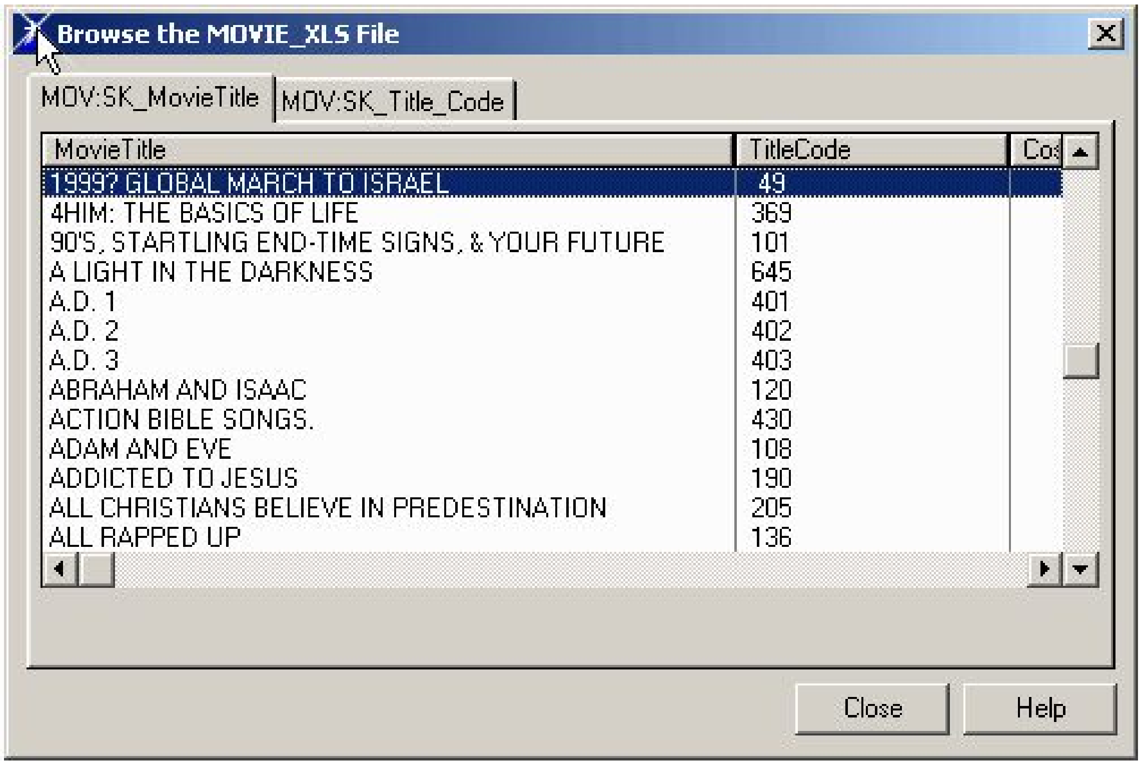
```
MOVIE_XLS            FILE,DRIVER('MSSQL'),|
                           OWNER('MSSQL:ConnectionString'),|
                                                   NAME('dbo.MOVIE_XLS'),PRE(MOV),|
                                                   BINDABLE,THREAD
SK_MovieTitle            KEY(MOV:MovieTitle),DUP,NOCASE,OPT
SK_Title_Code            KEY(MOV:TitleCode),DUP,NOCASE,OPT
Record                   RECORD,PRE()
```

```
MovieTitle                  CSTRING(256)
TitleCode                   CSTRING(256)
Cost                        CSTRING(256)
MovieDate                   CSTRING(256)
                         END
                     END
```

Step 5: Create a Browse.

Use the Browse template wizard to create a browse. Remember to make it view-only; do not attach an update form. To update data coming from a linked server, you need to use an SQL command like `OPENQUERY`. If you try to update the normal way, you will get an error.

Figure 3 shows what my created browse looks like, and it works just like any Clarion browse.



MovieTitle	TitleCode	Cos
1999? GLOBAL MARCH TO ISRAEL	49	
4HIM: THE BASICS OF LIFE	369	
90'S, STARTLING END-TIME SIGNS, & YOUR FUTURE	101	
A LIGHT IN THE DARKNESS	645	
A.D. 1	401	
A.D. 2	402	
A.D. 3	403	
ABRAHAM AND ISAAC	120	
ACTION BIBLE SONGS.	430	
ADAM AND EVE	108	
ADDICTED TO JESUS	190	
ALL CHRISTIANS BELIEVE IN PREDESTINATION	205	
ALL RAPPED UP	136	

Figure 3. Browsing an Excel Spreadsheet

As you can see, it is easy to view an Excel spreadsheet using a Clarion browse. This goes for any file structure that you add as a linked server, even a text file.

Ayo Ogundahunsi presently lives in Henderson, Nevada, about ten minutes from Las Vegas. He works for Impac Medical Systems Inc., the leading company in cancer therapy software (written in Clarion). Impac has its headquarters in Mountain View, California. Ayo is married to Ayodola, and they have two boys, Darren and Joshua.

Reader Comments

Add a comment

Clarion Magazine

Countdown To CLARION 6 Early Access Release

Published 2002-12-09

Welcome to the Clarion 6.0 Beta Watch. No, it's not something you wear on your wrist, but you might want to check it just about as often. On this page Clarion Magazine will track all the latest news on the upcoming Clarion 6.0 beta. Information is in date order, so the most recent information is at the end of this document.

December 6, 2002: From a newsgroup posting by Bob Zaunere

> Initial shipments of an Early Access (EA) release of Clarion 6 are expected to begin during the week of December 16th. And yes, it will be Clarion 6 not 5.6, as this release was deemed to have too many features for a dot release. All documentation of the new features is being finalized and assembled into PDFs and Help files, and when completed will be available on the SoftVelocity web site. The EA program is not a mass beta - participation is limited, and the program is targeted at third party developers and advanced developers who can provide detailed bug reports.
>
> The **focus** for the EA release is for **compatibility testing**, and as such we want developers to focus on using and learning the **new thread model**, and **testing of existing applications**.
>
> To get an EA copy of Clarion 6, developers must purchase either an **upgrade or new license** of Clarion 6. Pricing for Clarion 6 will remain the same as for Clarion 5.5 for both upgrades and new licenses.
>
> Clarion 6 is substantially complete; however the initial release will not include all

of the new technology and components.

The initial release will include:

- Runtime libraries implementing the new thread model
- IDE with many productivity enhancements
- Updated ABC, Clarion and ADO Templates
- Other new templates
- Updated ABC libraries
- XML class and parser support
- Updated drivers implementing the new thread model, and many enhancements
- New ADO support
- Support for XP manifests
- Completely rewritten 32bit Help file
- Documentation of new features in PDF format.

Evidently there are some other goodies to follow. EA participants will have access to a private forum. There will also be **live chat sessions** with SV developers and product managers, and **online training material** covering aspects of the new thread model, and other topics made available at no additional cost. **Contact SoftVelocity** if you wish to participate in the EA program.

SoftVelocity, Inc.
2769 East Atlantic Boulevard
Pompano Beach, FL 33062
Phone: 954-785-4555
Sales : 877-733-4555
Fax : 954-946-1650

Sales: sales@softvelocity.com
General Information : info@softvelocity.com

December 7, 2002: From a newsgroup posting by Bob Zaunere

Overseas customers can contact their local distributor. All distributors can deliver for the EA program.

December 7, 2002: From a newsgroup posting by Bob Zaunere

Mouse wheel support is in the RTL, so the mouse wheel works for either template chain, or without any template.

General release is expected in Q1 2003.

Support for **XP Manifests** is provided in the project system to link in a manifest resource. Bob says he hasn't seen any shading issues with Message() dialogs.

December 8, 2002: From a newsgroup posting by Bob Zaunere

The **documentation** is being completed, and will be posted before the EA release ships. The documentation is a work in progress, so as time progresses there will be additional materials, and updates to initial documentation.

December 9, 2002: From a newsgroup posting by Bob Zaunere

The EA release for C6 is targeted at **compatibility testing** of the platform, and to identify and resolve any migration issues. As such it won't include all new features, and you should not request access for the purpose of looking at new features.

December 9, 2002: From a newsgroup posting by Bob Brooker

SoftVelocity is monitoring the EA **participation requests** and will close entrance into the program when (or presumably before – ed.) the number becomes unmanageable.

December 9, 2002: Upgrade Price

Upgrading Clarion 5.5 Enterprise Edition to Clarion 6 **Enterprise** Edition: US$799

Upgrading Clarion 5.5 Enterprise *or* Professional Edition to Clarion 6 **Professional** Edition: US$350

December 9, 2002: From a newsgroup posting by Scott Ferrett

> In the 16bit C6 IDE you can press Alt-F2 on any entry control and get an entry dialog.

December 10, 2002: From a newsgroup posting by Bob Brooker

> "The final feature set of Clarion 6 has not been established yet... not all subsystems are scheduled to be included in the EA, it is primarily for platform, migration and compatibility testing. It is not [SoftVelocity's] intent to provide the EA program as a mechanism to get a "sneak peak" at new features. Specific features intended for Clarion 6 when it ships will be discussed in [SoftVelocity's] newsletters, website and product specification sheets."

December 10, 2002: From a newsgroup posting by Bob Brooker

> "[SoftVelocity has] not stated that Clarion 6 user interface controls are fully XP "themeable"… Clarion 6 has XP manifest support and works within the parameters of that support."

December 10, 2002: From a newsgroup posting by Bob Brooker

> "There are some minor template changes required [to make Clarion/ASP compatible with Clarion 6] and there will be an update
> for the Clarion/ASP template set prior/concurrent to Clarion 6 being released. The templates work with Clarion 6 as they do in Clarion 5.5."

Reader Comments

Add a comment

Clarion News

Published 2001-11-21

SealSoft xAnalogClock 1.2

xAnalogClock v1.2 is now available. This is a bugfix release. A new demo and install are now available. The new Install Kits password will be emailed to registered users.

Posted Friday, December 20, 2002

Clarion 6 EA Program Now Full

According to the Netherlands Clarion distributor, the Clarion 6.0 Early Access Program is now full, and no more applications are being accepted.

Posted Friday, December 20, 2002

Another RADrace Victory For Clarion!

The star-race-team of RADventure, Erik Pepping and Peter Rakke, has succeeded in winning the prestigious RADrace two years in a row, an unprecedented accomplishment. They used Clarion 5.5 Enterprise Edition in combination with RADventure tools. The RADrace is an yearly event where teams of developers compete to complete as much as possible of a real life business case application within a very limited timeframe (2 business days). Part of the challenge is the inevitable change-of-mind of the customer requirements at a very late stage. This years assignment was a European museum ticket and access program; requirements included business rules, email and a web access interface.

Posted Friday, December 20, 2002

File Manager 3 Beta 9a

CapeSoft's File Manager 3 beta 9a is now available for download. ODBC users will be pleased to know that there is now "first release" support for MySQL and Oracle through ODBC. Please send your comments, suggestions, queries, and bug reports to fm3@capesoft.com.

Posted Friday, December 20, 2002

New ImageEx 2 Demo

A new demo of ImageEx2 is now available. This version includes an animated rotating cube and a box-cover editor.

Posted Friday, December 20, 2002

ConVic 2003

Yes, there will be another ConVic this year. ConVic 2003, an Australian Clarion developers' conference, will be held in Geelong, Victoria, March 28-30, 2003. The conference will be accompanied by a two-day training course titled Using Clarion for Client/Server Database Development.

Posted Friday, December 20, 2002

CPCS Christmas Schedule

CPCS will be closed for Christmas from Sat. Dec. 21, 2002 till Sun. Dec. 29, 2002. All email and newsgroup messages received during that period will be handled as soon as possible after Dec. 29, 2002.

Posted Monday, December 16, 2002

SealSoft New Year Discount

SealSoft is offering a New Year's discount on all products purchased from December 15, 2002 till January 7, 2003. The discount is equal to 20% + discount by personal discount card.

Posted Monday, December 16, 2002

wPDFControl Wrapper

A Clarion wrapper for the wPDFControl DLL from wpcubed GmbH (www.wpcubed.com) is now available from Klarisoft. The wPDFControl DLL lets you create full featured PDF files directly from Clarion application. Built in WMF support means you can save Clarion reports directly to PDF files. The DLL also supports direct drawing to PDF file, graphic primitives, true type fonts, images, bookmarks, outlines, hyperlinks, password protecting, and compression. No need for the ActiveX registration as the wrapper works directly with the DLL. A trial version of the wPDFControl DLL is available from www.wpcubed.com. The wrapper is available as a source or compiled DLL version and works with both Legacy and ABC template chains.

Posted Monday, December 16, 2002

New Icetips Bulletin Board

IceTips has set up a new bulletin board. The old board is now closed and all old messages have been removed. The IceTips board is a service for all Clarion developers, and IceTips will also use it to communicate with beta testers in the future rather than by email.

Posted Monday, December 16, 2002

Clarion Template/API Forum

Roel Abspoel is setting up a forum with template and API downloads for Clarion. The forum is free. Mainly it is a place where Roel keeps some references, but feel free to post your own templates and API info if you like.

Posted Monday, December 16, 2002

Clarion Source Code For Sale

Tiger Programs is offering the Tiger SIR application's source code for sale. This application is designed to run with a PC with touch screen and Pocket PC's. It's fully supported by Clarion 5 and 5.5 using the Terminal Server services of the Pocket PC. This sale is contingent on a minimum of 50 buyers. The total cost of the application with all current versions is US$1,600

Posted Monday, December 16, 2002

Clarionfoundry Open To Public Again

You no longer need a login and password to access Clarionfoundry. There is also a new section in Clarionfoundry called Clarion6, which will be addressing the various deployment, conversion, new features etc.

Posted Monday, December 16, 2002

IceTips December Newsletter

The IceTips December newsletter is now available. It is completely free and has information about current IceTips projects, Clarion 6 compatible upgrades, and some technical tips as well.

Posted Monday, December 16, 2002

Icetips Holiday Specials

Icetips Software is running a 25% special Holiday Sales on selected items, the Icetips Previewer, Icetips Magic Buttons and Icetips Magic Entries. Icetips Previewer, now $149, save $50; Icetips Magic Buttons, now $59, save $20; Icetips Magic Entries, now $59, save $20; Icetips Magic Bundle, now $93, save $32. Purchase the Previewer and the Magic bundle for $242 and save $82. Upgrades to Clarion 6 compatible versions are free of charge.

Posted Monday, December 16, 2002

Possible Florida UG Conference

Please let Mark Goldberg know if you would be interested in attending a Clarion Developers Meeting in Orlando. This event is currently in the early planning stage, and would be for members of all Florida Clarion User Group members, although anyone may participate. To make this easier for all, the event would be held at the Orlando Airport hotel (site of one of the Devcons in the past), possibly in February 2003. The initial idea is to hold this on a Friday/Saturday type deal, with a nice social meet and greet dinner after the first day. Then Saturday will be about a half day (maybe a full one).

Posted Monday, December 16, 2002

ImageEx2 Beta 3

Beta 3 of ImageEx 2 is now available. The installation password has not changed, so please use the one that was sent to you on purchase. New features include: Clickable hotspots (rectangles, ellipses & polygons) for the viewer control; Screen capture functions for capturing the entire desktop, single windows or rectangles; Improved PictureDialog. See the online documentation for a complete list of changes. A new demo is also available. This will be the last beta version of ImageEx, so this might be your last chance to save $50 on purchase, as the price will go up to $199 with the gold release.

Posted Monday, December 16, 2002

S.C.A. Micro Legacy Templates

As requested, there is a new template set for Clarion/Legacy templates Pricing is the same as the ABC templates: $15. You can buy both for $22.50.

Posted Monday, December 16, 2002

ClarionPost Reopens

After a few months of restructuring, www.clarionpost.com is again open. ClarionPost.com provides all the third party Clarion developers a centralized location to enter their links

Posted Monday, December 16, 2002

Last Chance For List & Label Discount

Combit have said the new version of List & Label (version 9) will be released next Wednesday. Orders taken before then for either upgrades or new licenses will be discounted by $50. Simon Burrows also reports that the RTF object that is in the layout designer is available as a separate control, so you 'should' be able to include this as a separate RTF control within your Clarion app.

Posted Friday, December 06, 2002

Simsoft Christmas Stocking - Save $58US

Until the end of December 2002 you can now obtain the Simsoft Christmas Stocking for just 109$US. The stocking is filled with: Simsoft Templates (normal price $69US); Simshape Templates (normal price $49US); Simpad Templates (normal price $49US). Existing users can upgrade from existing bundles for the difference in price.

Posted Friday, December 06, 2002

International Clarion Meetup Day

The first International Clarion Meetup Day will be January, 14th at 7:00pm local time everywhere. Meetup creates real-world group gatherings for almost 80,000 people right now, about anything anywhere. Meetup has built a technology and a network of venues (cafes, bars, etc.) that can help any interest group easily organize local monthly meetups in over 530 cities across 27 countries.

Posted Friday, December 06, 2002

Save 10% On Image Man OCX

Data Techniques is offering a 10% discount on their ImageMan controls. These are the imaging controls that The Nextage ImageMan templates are built around.

Posted Wednesday, December 04, 2002

RInstall V1.f (Beta) Update

The RInstall template and application version 1.f (Beta) has been updated. This release allows you to disable certain API calls which can cause a clash with other templates calling the same API calls.

Posted Wednesday, December 04, 2002

INN Bio & News for 3-Dec-2002

This week, the Icetips News Network is pleased to feature a rather well-known Clarion programmer who claims "I don't do software development". Riiiight... he also says, "data is executed policy and therefore the definition of data is the essence of corporate policy". Hmmm. Well, don't think it's all serious, Gramps manages to make fun of quite a few things along the way, including Arnor's hairline.

Posted Wednesday, December 04, 2002

Clarion Handy Tools Newsgroup Server

On December 2, 2002 The Clarion Handy Tools Page released its O7B2.0 build to subscribers. This build includes significant advancements in a number of areas such as SMTP email, Browser Server, SQL support, Encryption and Compression support and more. This is a "pre-final" version of the O7B2.0 build because it was released a couple of weeks ahead of the scheduled Dec 15th date. Final "What's New" docs for the build will be posted in the form of an update on or about Dec 15th. One of the example applications included with the O7B2.0 build kit is a full-fledged newsgroup server built with Clarion and The Clarion Handy Tools. You can use it as the starting point for your own newsgroup server, since the source application is available in your tool kit. This server is running and available for CHT support. The on-line help explains how to set up a desktop icon that gives you instant access to the latest messages using an auto-login string.

Posted Wednesday, December 04, 2002

Shapemaker SMX

Logic*Central has released Shapemaker SMX, an new product that applies the Shapemaker technology to buttons. You can design your button with the SMX Designer (like the Shapemaker Polydesigner) and compile it in Clarion. It is not an OCX, but all-Clarion code

with these features among others: ButtonColor; Text; Icons; Alignment; Text and Icon effects; Text Styles; Text fonts. Template and SMX designer will be ready for sale at the end of the year or in the first week in January 2003.
Posted Wednesday, December 04, 2002

Subject: Nextage Imaging Update

New versions of both the Imagining Templates and the ImageMan Templates are now available. Changes include: Bug fix for compile problems if you do not populate the thumbnail template in your application; Bug fix for a problem with print procedure control not including the include file; ADF scanning bug fix.
Posted Wednesday, December 04, 2002

xFText v2.0 Released

xFText 2.0 is now available. This is the advanced version, with changes to the templates and class methods. There is no black box DLL - everything is Clarion code and WinAPI calls. Supports single exe, multi Dll (Local Mode, Standalone Mode), 32-bit. Features include:; Set global margins for text; Set all attributes of font. Name, size, style, color, charset; Write text into any place of Frame. Left-Top, Center-Top, Right-Center etc.; Set "Normal", "Light" and "Dark" color of text for 3D text imitation; Set offset for X- and Y-position; Set offset for LightColor and DarkColor for 3D texts; Add, change and remove frame text in runtime; Add bitmap image on Frame; Set all parameters via variables. New demo and install kit available. Attention registered users - install password was changed and will be sent to you via email.
Posted Wednesday, December 04, 2002

DOS Printer v7.4

David Beggs has released DOS Printer 7.4. This utility sits in the system tray waiting for a specified file (wildcards ok) to exist. When that file exists, DOS Printer converts it to a windows report format and prints it to any Windows printer or printer driver you like. You can also pick a file and print it. DOS Printer can also email the file (including its formatting), or print it to a fax driver, PDF maker or any other sort of Windows printer driver. Price $US19.99. Developer (i.e. distribution) licensing available on request.
Posted Wednesday, December 04, 2002

BigTamer Update

A new release of the BigTamer templates, with new pricing, is now available.
Posted Wednesday, December 04, 2002

ZipApp Free Backup Program

This free utility from Darron Pitman does recursive zips of your development files, creating new zips each time rather than overwriting.
Posted Wednesday, December 04, 2002

Reader Comments

Add a comment

Clarion Magazine

Data Structures and Algorithms Part XIII - Trie Hard

by Alison Neal

Published 2002-12-19

In my last article I introduced a data structure called the Trie, which is used for storing strings in computer memory. In this article I will continue my discussion of the Trie, and cover some of the other methods that are contained in the `Trie` class, namely the `Search`, `Pr` (print) and `Kill` methods.

The Search Method

Below is the code I have written for the Search method:

```
Trie.Search                PROCEDURE(*STRING s)
ch  STRING(1)
m   ULONG(1)
i   ULONG(0)
q   &bNode
t   &dNode
r   LONG(0)
    CODE

    IF SELF.root = 0 THEN RETURN FALSE.

    r = SELF.root

    LOOP WHILE r <> 0 AND m <= LEN(CLIP(s))
      q &= (r)
      IF q.n <> 0 THEN BREAK.
      ch = UPPER(s[m])
      m+=1
      i = SELF.pos(ch)
      IF i < 1 OR i > nBranches THEN RETURN FALSE.
      r = q.p[i]
    END
    IF r = 0 THEN RETURN FALSE.
    t &= (r)
    RETURN CHOOSE(CLIP(t.s) = CLIP(UPPER(s)), TRUE, FALSE)
```

The purpose of the Search method is to only tell me whether a word exists in the Trie structure by returning either true or false. Looking at the example from the previous article (Figure 1.) let’s assume that I want to find the word "IMAGINE".

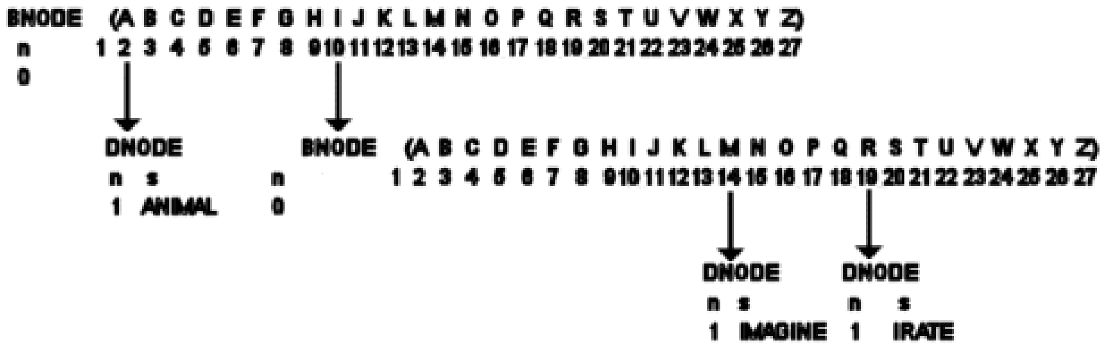

BNODE (A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)
n 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
0
DNODE
n s
1 ANIMAL
BNODE
n
0
(A B C D E F G H I J K L M N O P Q R S T U V W X Y Z)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27
DNODE
n s
1 IMAGINE
DNODE
n s
1 IRATE

Figure 1. The trie from Part 1.

If there was nothing in the Trie, and the root (`LONG`) was still zero – i.e. didn’t contain a memory address - then the `Search` method would automatically return false. However this is not the case in this example, so r is made to equal the root which contains the memory address for the root Branch Node.

The structure is then looped through, until such a stage as r no longer contains a valid memory address, or the string character position keeper (`m`) is higher than the length of the string. Next, `q` is made to refer to the node whose memory address is stored in `r` (the root to start). Since the `n` variable tells me whether the node type is a Branch Node (`bNode`) or Data Node (`dNode`), a check is made to ensure that the current node is a Branch Node, that is that the n value is equal to zero. If it is not, then the search has reached the lowest point in the structure that it can go to and must break out of the loop. In this example the root node does have an n value of zero, because it is a Branch Node.

At this stage the first character of the search string is analysed to find out where in the array the process should look next. The search string is "IMAGINE", its first character is "I", and therefore the position in the array that the process should look to is 10. `I = 10`, which is not less than one, and not greater then the `nBranches` equate. This check makes sure that the search doesn’t overrun the bounds of the array, which is defined to the size of the number of letters in the alphabet.

Now `r` is made to equal the memory address that is stored in the 10th position of the array, and the loop cycles. If nothing were stored in this position then the loop would terminate, as r would equal 0. In the example this memory address refers to a second Branch Node, as shown in Figure 1.

This time through the loop n still equals 0, as this is a Branch Node, and the second letter in the search string is "M", which is position 14 in the array. So `r` now holds the memory address of the Data Node containing the word "IMAGINE".

In the next iteration the n value is 1, and therefore the process breaks out of the loop. A check is made directly after the loop to make sure that the process hasn't reached a natural end, meaning `r` equals 0 and therefore does not contain a memory address. If this were the case then false would be returned.

As `r` does not equal zero it must be assumed that the node presented must contain a string, and that string is the closest match to the word being searched for in the Trie. If the strings are a complete match then the process returns true, otherwise false.

This is a very simple search function, which could be expanded on quite significantly. For example, a useful search function could return a list of all words that start with the letters passed to it, or it could return a list of words that the search term closely (like a spell checker).

The Pr Method

Here is the code I have written for the print method:

```
Trie.pr               PROCEDURE()
    CODE
    CREATE(ExportFile)
    OPEN(ExportFile)
?   ASSERT(~ERRORCODE())
    SELF.prnt(SELF.root)
    CLOSE(ExportFile)
Trie.prnt               PROCEDURE(LONG r)
i   ULONG
t   &dNode
q   &bNode
    CODE
    IF r
      t &= (r)
      IF t.n
        Exp:Line = t.n &' ' &t.s
        ADD(ExportFile)
      ELSE
        q &= (r)
        LOOP i = 1 TO nBranches
          IF q.p[i] THEN SELF.prnt(q.p[i]).
        END
      END
    END
```

The Pr method, like the Search method, traverses the Trie, but unlike the Search method it visits every Node. Here's how the code works, using the example given in Figure 1.

The `pr` method creates and opens the export file and then calls the `prnt` method, passing the memory address stored in the root. As the root does contain a memory address, `t` is made to reference that node. A check is then performed to see what type of node this is. In this instance it is a branch node, so `q` (`bNode` type) is made to refer to the node, and each of the array elements are looked at. In the example, position one contains nothing, so on the first recurse nothing happens. The check is made for `r` – the memory address – and as this is zero, the procedure just ends without doing anything and returns to the original calling instance of the `prnt` procedure.

Position two contains the address of the data node containing the string "ANIMAL", so on the second recurse `t.n` is true, and thus the word and the count (`n`) of the word occurrences are added to the export file. The procedure then returns to the original caller, and the loop continues making a recursive call each time and passing a zero value, until reaching the tenth position, which contains the memory address for the second branch node.

As `r` is not zero and `n` is zero, the loop is executed once more, this time looping through the second branch node. On reaching positions 14 and 19, the code adds "IMAGINE" and "IRATE" to the export file. The loop then continues to the last position, and the procedure terminates, returning to the loop of the original branch node. The first loop then continues to the end of the array, with nothing more to add to the export file and the procedure terminates successfully.

The Kill Method

The code I have written for the Kill method is as follows:

```
Trie.Kill               PROCEDURE()
    CODE
    SELF.rem(SELF.root)
Trie.rem               PROCEDURE(LONG r)
t   &dNode
q   &bNode
i   ULONG(0)
    CODE
    IF r
      t &= (r)
      IF t.n = 0
        q &= (r)
        LOOP i = 1 TO nBranches
          IF q.p[i] THEN SELF.rem(q.p[i]).
        END
      END
      DISPOSE(t)
    END
```

The `Kill` method, like the `Pr` method, traverses the Trie, visiting every node in the structure. This method works in a very similar way to the `Pr` method inasmuch as it checks to see if a node is valid, then determines what type of node it is. If the node is a branch node type, then the procedure loops through the array and makes a recursive call, passing the value stored in the array position as the next node of relevance. If the branch node has already been checked, or the node is a data node, then it will `DISPOSE` of the current node.

Summary

The methods associated with the Trie structure could be expanded extensively to perform some really useful functions. For instance, with a little modification the Search method can return all words that contain specific letters, or are made up of particular letters. In this way the Trie can be used as a spell checker, to return those words, which are similar to the word to be checked. Its uses are countless. The most common use of a type of Trie is in Huffman's file compression algorithm. This algorithm however also requires the use of a structure known as the Priority Queue, so in my next article I will provide some insight into what a Queue actually is, and how it's supposed to work. That will lay the groundwork for the Priority Queue structure, and Huffman's file compression algorithm.

Alison Neal has been using Clarion since 2000, whilst working for Asset Information Systems (AIS) in Auckland, New Zealand. Some years ago (at the tender age of 19) Alison graduated from the Central Institute of Technology in Wellington, New Zealand with a major in Cobol. She also has a BA in English literature and has studied Computer Science, Philosophy and Information Systems. AIS is an independent division of Asset Forestry Ltd, and has a team of five programmers developing almost exclusively in Clarion. AIS also offers web (ClarioNET) and email services for the customer who needs everything. The company has many and varied customers bridging across a wide range of industries including Telecommunications, Forestry & Agriculture, Manufacturers, Military & Government, Legal & Financial, and Retail.

Reader Comments

Add a comment

Clarion Magazine

For marketing your Applications
and Developer Accessories or to
purchase other 3rd Party Tools . . .
Developer PLUS
tm

Ads · Comments · Writers! · Privacy · Contacts · PDFs · Freebies · Open Source

Topics > Tips/Techniques > Clarion Language

DNA for Clarion: Manipulating Browse Cells With A VLBPROC (Part 1)

by Stephen Bottomley

Published 2002-12-19

It seems there has been an increased interest in virtual list boxes, what they can be used for, how to use them and how to create a template for them. While this article won’t be the definitive answer, I hope it will help answer a few of those questions and give you some idea as to whether a virtual list box might be of use to you.

What is a Virtual List Box?

A virtual list box is a mechanism whereby you can communicate with a list control without using a queue. Unlike using the physical structure of the queue to define and store data and property information a virtual list box uses a callback function to virtualize or mimic the functionality of the queue to determine how many rows and columns to display and the data and property information to be shown in each cell.

How does it work?

First you populate a list box control on the window, the `FROM` attribute is optional depending on how you will use the virtual list. Then you will use the two runtime properties `PROP:VLBval` and `PROP:VLBproc` to tell the list box the address of the class that it will be using to supply information in place of a queue (`PROP:VLBval`) and the address of the callback function (which will actually be a method of the `PROP:VLBval` class) that it will call to mimic the information usually supplied by a queue(`PROP:VLBproc`).

Once `PROP:VLBval` and `PROP:VLBproc` have been assigned, any queue used in the `FROM` attribute will be ignored. The list box will *only* call the `VLBPROC` to request the different types of information, number of rows, number of columns, is there a need to change the data currently displayed, and lastly, for each cell, what data should be displayed.

The warning

While not VLBPROC specific it should be noted that in versions prior to C5.5E there was a bug that if the program changed PROPLIST:Color or other properties that require queue field(s) (which implicitly includes calls to a VLBPROC for those fields) for storing values at run time for (<not last> and <not first>) column, the RTL builds the temporary format string incorrectly.

The basics

What you need is a class that will fulfill all the requirements of a virtual list box but it should also be designed in such a way as to be useful for different purposes, a base class. Let’s take a look at how this might be constructed.

```
JtCmDnaClass CLASS,TYPE
Feq            SIGNED,PROTECTED
Window         &Window
Changed        PROCEDURE,BYTE,VIRTUAL
Cols           PROCEDURE,SHORT,VIRTUAL
Init           PROCEDURE(WINDOW w,SIGNED Feq)
Rows           PROCEDURE,LONG,VIRTUAL
Splice         PROCEDURE(LONG Row,SHORT Col),STRING,VIRTUAL
VlbProc        PROCEDURE(LONG Row,SHORT Col),STRING,PRIVATE
             END
```

As you can see, we have defined a base class. In it’s own right it won’t do a great deal but creates the platform from which the contents of each cell in your list box can be manipulated. The base class contains a holder for the list boxes Field Equate (FEQ), a reference to the window that the list box is on(Window), an Init method and a generic VLBPROC callback method. The other four methods are all virtual and will be used by any derived class to do the actual work of defining list box properties and the makeup of the cells.

Initialization

The Init method handles both the initialization of the two class properties as well as setting the two list box properties with the required information it needs in order to become a virtual list box. The VLBPROC method calls the appropriate virtual method in response to the value of the Row parameter. –1 for the number of rows the list contains, -2 for the number of columns, -3 to let the list box know if anything has changed and the list should be re-filled. Anything else in the Row parameter indicates that you should return the actual data to splice into the cell at the row and column numbers specified in the parameters.

```
JtCmDnaClass.Init PROCEDURE(WINDOW w,SIGNED Feq)
```

```
  CODE
  SELF.Feq = Feq
  SELF.Window &= w
  SELF.Window $ SELF.Feq{PROP:VLBval} = address(SELF)
  SELF.Window $ SELF.Feq{PROP:VLBproc} = address(SELF.VLBproc)
JtCmDnaClass.VlbProc PROCEDURE(LONG Row,SHORT Col)
  CODE
  case row
  of -1
    return SELF.Rows()
  of -2
    return SELF.Cols()
  OF -3
    return SELF.Changed()
  else
    return SELF.Splice(Row,Col)
  end
```

To complete the base class the default code for the four virtual methods needs to be filled in:

```
JtCmDnaClass.Changed PROCEDURE
  CODE
  return 0
JtCmDnaClass.Cols PROCEDURE
  CODE
  return 0
JtDnaClass.Rows PROCEDURE
  CODE
  return 0
JtCmDnaClass.Splice PROCEDURE(LONG Row,SHORT Col)
 CODE
  return('')
```

What now?

Now you need to find a use for a virtual list box that either can't be better handled by a queue or, a list box that is currently handled by a queue but could gain some benefit from additional information.

One idea springs to mind. That is to make a single class that can extend the display properties of list boxes created using different types of browse templates (a mix of standard, legacy and/or third party browse templates)

Gene Modification 101: Let's build a Zebra

Like Virtual List Boxes, another subject that comes up from time to time in the news groups is the ability to greenbar a browse. Greenbarring is where each line of a list alternates between two colour sets in a Zebra stripe effect (see Figure 1). I've created a template-based solution that is freely available from my website but there are two major drawback with the template-only approach. First, it can only be used for specific browse type. I had to build a completely different template to do the same thing for the Clarion browse template as well as the ABC browse and would need another one for each type of browse template I used. Drawback two, none of what I had created could be used for standard list controls.

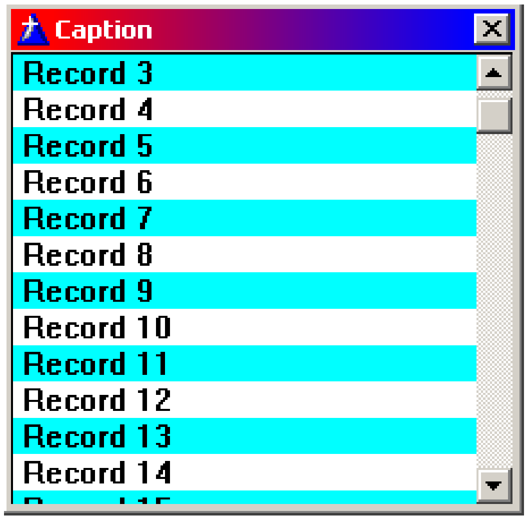


Figure 1. A list with a greenbar effect.

What if you could create a means to greenbar a list that is template and code independent? Would that be nice?

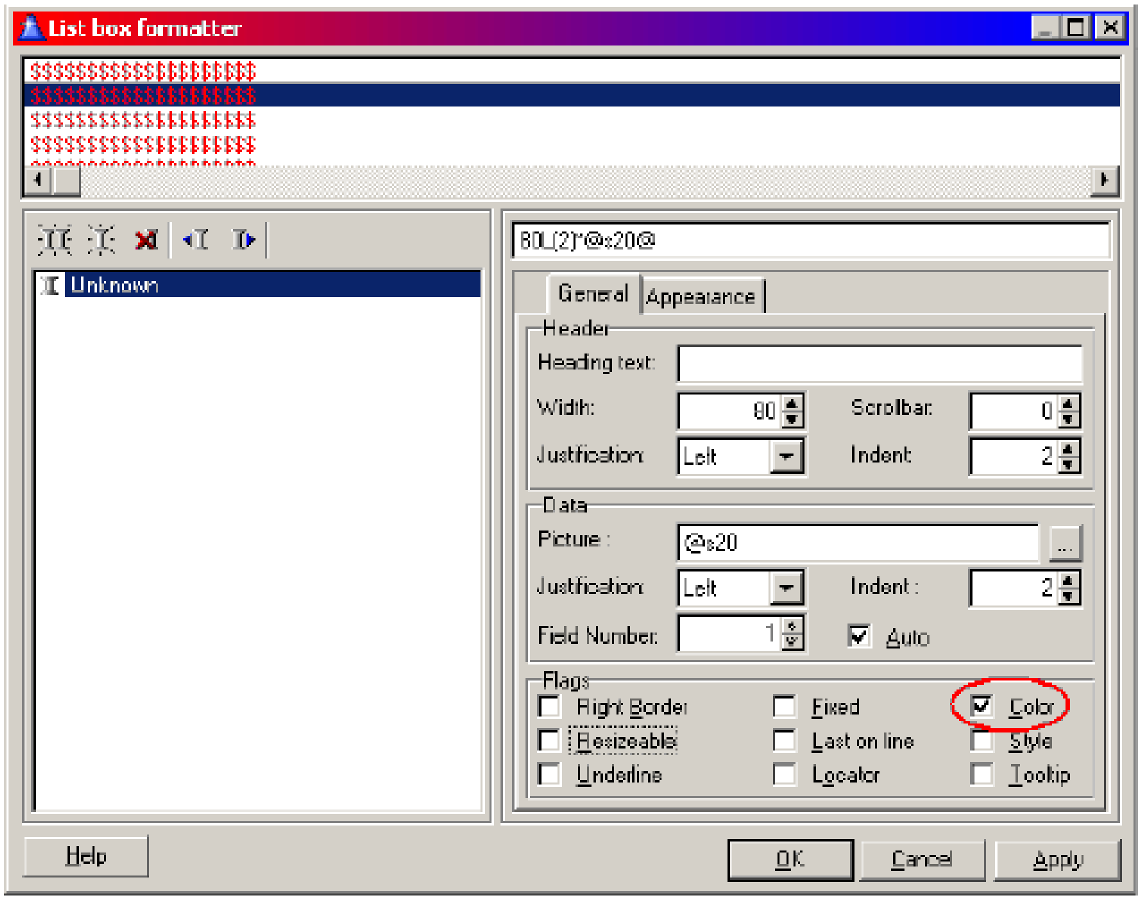
The considerations

With this manipulation of the gene pool your virtual list will be taking over control of a list box that is being used by other code using a queue to display information. The result should be able to be used independent of template chain considerations or even independent of template generated code all together. The one exception of course is where a VLB is already being used.

The possibilities for features are extensive but let’s stick to a fairly basic Zebra. What we are going to do is use the `VLBProc` to "uncouple" the original queue from the list box control and than act as a conduit between the queue and the list control to pass the original display data and also splice into that stream the information required to add the stripe effect to the list.

The Zebra class

Just like a list box with a queue, the list box format must have the `COLOR` attribute set for each data column that will be colored. Also, the virtual column structure of your Zebra class will still have to know which columns are the colour columns. Your Zebra also needs to know what colour each of its stripes will be. Set the Color check box as shown in Figure 2.


List box formatter

80L(2)~@s20@

General | Appearance

Header
Heading text:
Width: 80
Scrollbar: 0
Justification: Left
Indent: 2

Data
Picture : @s20
Justification: Left
Indent : 2
Field Number: 1
[✓ Auto]

Flags
Right Border, Fixed, [✓ Color]
Resizeable, Last on line, Style
Underline, Locator, Tooltip

Figure 2. Setting the color attribute for a column

To do this you can use a queue to store the column number and colour options of the colour columns. Each element will contain the column number that it acts upon plus the normal and stripe colour. Then create the class definition by inheriting the DNA base class and adding the extra properties and methods required.

```
JtCmDnaZebraColourOptionQ QUEUE,TYPE
Col                         SHORT
Normal                      LONG
Stripe                      LONG
                          END


JtCmDnaZebraClass  CLASS(JtCmDnaClass),TYPE
Changes              LONG,PRIVATE
DDQ                  &QUEUE,PRIVATE
Options              &JtCmDnaZebraColourOptionQ,PRIVATE
AddStripe            PROCEDURE(SHORT Col,LONG Normal = -1,LONG Stripe = -1)
Changed              PROCEDURE,BYTE,VIRTUAL
Cols                 PROCEDURE,SHORT,VIRTUAL
Construct            PROCEDURE,PRIVATE
Destruct             PROCEDURE,PRIVATE
IsColorCol           PROCEDURE(SHORT Col),BYTE,PROC,PRIVATE
Init                 PROCEDURE(WINDOW w,SIGNED Feq,QUEUE Ddq)
Rows                 PROCEDURE,LONG,VIRTUAL
Splice               PROCEDURE(LONG Row,SHORT Col),STRING,VIRTUAL
                   END
```

Additional properties and methods:

- Changes: Will be used to track change conditions of the default data queue.
- DDQ: A reference to the default data queue being used to display the list box data
- Options: The class implementation of the colour option queue
- AddStripe: Adds stripe information for each colour column
- Construct: Called automatically when the object is instantiated
- Destruct: Called automatically when the object is destroyed
- IsColorCol: Determines if a column is one of the colour columns
- Init: Overloads the base class Init method with one extra parameter

The remaining four virtual methods, `Changed`, `Cols`, `Rows` and `Splice` will also contain Zebra specific code.

The cloning

The Zebra virtual list box will be an enhanced clone of the default list box. To this end it will need to know the structure it is being created from. This is done using the `Init`, `Changed`, `Cols`, and `Rows` methods.

The `Init` method initializes the `DDQ` property with a reference to the default data queue (remembering that once the `VLBPROC` has been initialised the list box will ignore the queue originally assigned to the `FROM` attribute). It calls the Init method from the base class and then initializes the change state of the default data queue.

```
JtCmDnaZebraClass.Init PROCEDURE(WINDOW w,SIGNED Feq,QUEUE Ddq)
  CODE
  SELF.DDQ &= Ddq
  SELF.Init(w,Feq)
  SELF.Changes = changes(SELF.DDQ)
```

The `Changed` method will track differences in the default data queue to determine if the list display needs to be updated.

```
JtCmDnaZebraClass.Changed PROCEDURE
Changes LONG
  CODE
  Changes = changes(SELF.DDQ)
  if Changes <> SELF.Changes
    SELF.Changes = Changes
    return 1
  else
    return 0
  end
```

The `Cols` method counts the number of columns displayed in the list box.

```
JtCmDnaZebraClass.Cols PROCEDURE
Counter  SHORT,AUTO
  CODE
  Counter = 0
  loop
    Counter += 1
    if not SELF.Feq{PROPLIST:Exists,Counter}
      Counter -= 1
      break
    end
  end
  return(Counter)
```

The `Rows` method returns the number of records currently loaded in the default data queue.

```
JtCmDnaZebraClass.Rows PROCEDURE
  CODE
  return(records(SELF.DDQ))
```

The enhancement

The enhancements are created through the `AddStripe` method. This method is called to add or change stripe information for each colour column. Every entry in the option queue will contain the associated colour column number as well as the normal and stripe colour information. There will be four entries for each displayed column, just like a normal list box using a queue (normal foreground, normal background, selected foreground, selected background).

```
JtCmDnaZebraClass.AddStripe PROCEDURE(SHORT Col,|
                             LONG Normal = -1,LONG Stripe = -1)
PutRec  BYTE
  CODE
  PutRec = SELF. IsColorCol (Col)
  SELF.Options.Col      = Col
  SELF.Options.Normal   = Normal
  SELF.Options.Stripe = Stripe
  if PutRec
    put(SELF.Options)
  else
    add(SELF.Options,+SELF.Options.Col)
  end
```

Gene splicing

The last two methods are used to manipulate the information that will be stored in each cell of the enhanced clone.

`IsColorCol` detects if the information being requested by the list box is one of the Zebra stripe colour columns.

```
JtCmDnaZebraClass. IsColorCol PROCEDURE(SHORT Col)
  CODE
  SELF.Options.Col = Col
  get(SELF.Options,SELF.Options.Col)
  if errorcode()
    return(False)
  else
    return(True)
  end
```

The `Splice` method fills each cell with the required information. It does this by first synchronizing the default data queue with the requested row. It checks if the requested column is one of the Zebra stripe colour columns. If not, the data from the default data queue is returned to the cell. If it is, there is a check to see if same column in the default data queue contains a value other than the default `COLOUR:None`. If it is not the default colour then the default data queue colour is used. This allows any conditionally colored cells to be displayed as expected. Otherwise the splicing tests to see if the requested row is odd or even and replaces the default data queue colour with the appropriate Zebra colour from the option queue.

```
JtCmDnaZebraClass.Splice PROCEDURE(LONG Row,SHORT Col)
  CODE
  get(SELF.DDQ,Row)
  SELF.Options.Col = Col
  get(SELF.Options,SELF.Options.Col)
  if errorcode()
    return what(SELF.DDQ,Col)
  else
    return choose(what(SELF.DDQ,Col) <> -1|
          ,what(SELF.DDQ,Col),choose(not band(row,1)|
          ,SELF.Options.Normal,SELF.Options.Stripe))
  end
```

Bringing the Zebra to life

Like any genetic manipulation techniques, each step needs to be performed in the correct sequence at the right time. Fortunately for the Zebra, now that all the genetic information is complete, the steps to life are few and pretty simple.

First requirement is a queue loaded list box to clone from with at least one column set with the `COLOR` attribute. Next is an instance of the `Zebra` class, third is the initialization of the default Zebra information and lastly the addition of the Zebra stripe enhancement information.

Once the class information has been added to your project (usually by use of `INCLUDE` statements in the Global Data area of your project or application) creating an instance of the

Zebra class is straightforward. This should be done in the data section of the procedure that contains the list box you want to add stripes to.

```
Zebra     JtCmDnaZebraClass
```

Next you need to initialize the Zebra with the information from the list box it is going to clone. Because the Zebra class is going to use property assignments to take control of the list box then like any other property assignments, this must be done after the window is open. In the statement below, Window is the label of the window definition that contains the list box. ?Browse:5 is the USE variable of the list box that will be cloned. Queue:5 is the label of the default data queue assigned to the list box's FROM attribute.

```
Zebra.Init(Window,?Browse:5{PROP:Feq},Queue:5)
```

Lastly you need to determine which columns will have stripes and what the normal and striped colors will be. Each call to AddStripe will be for one of the four colour columns associated to each colored display column. Assuming Column 1 has the COLOR attribute set, to make the normal text green, normal background teal, selected text and background colors standard this would be:

```
Zebra.AddStripe(2,COLOR:Blue,COLOR:Yellow)
Zebra.AddStripe(3,COLOR:Aqua,COLOR:Teal)
Zebra.AddStripe(4,COLOR:Aqua,COLOR:Yellow)
Zebra.AddStripe(5,COLOR:Gray,COLOR:Navy)
```

You now have a living breathing Zebra, and what could be easier? The answer, for generated applications at least, would be if you didn't have to figure out which columns were the colour columns and apply the information manually to each one. Next time I'll show how that's done.

Download the source

Steve Bottomley is a long time user of Clarion, a member of Team Topspeed, and works for the Australian Government.

Reader Comments

Add a comment

Clarion Magazine

DNA for Clarion: Manipulating Browse Cells With A VLBPROC (Part 2)

by Stephen Bottomley

Published 2002-12-20

Last time I introduced a set of classes that use Clarion's Virtual List Box (VLB) capability to add greenbars to any ABC or Legacy browse. This week I'll add a set of templates that make these classes even easier to use, and I'll also demonstrate a handcoded use of the greenbar class.

Enter the template. Actually it's templates. In order to maintain the theme of template independence and also maximize hands free flexibility plus include some future proofing it will be three simple code templates. They are:

- The Global Template will add the `INCLUDE` statements.
- The Data Template will create an instance of the class.
- The Initialization Template will set all the colour columns with the selected colors.

The template header

If you're going to create the code templates as a standalone template set rather than incorporating them into a current chain you need to add the template header. This includes a unique label, a description and the template families it can be used in.

This header includes the ABC and CW20 template families and could be extended is you use your own template chain.

```
#TEMPLATE(JtCmDna,'(JaDuTech-CM) DNA for Clarion'),|
  FAMILY('ABC'),FAMILY('CW20')
```

The global template

This template has one prompt for the Zebra class at this stage and could be easily extended to include other classes that inherit the DNA base class. This template is placed in the Global Data embed of your application.

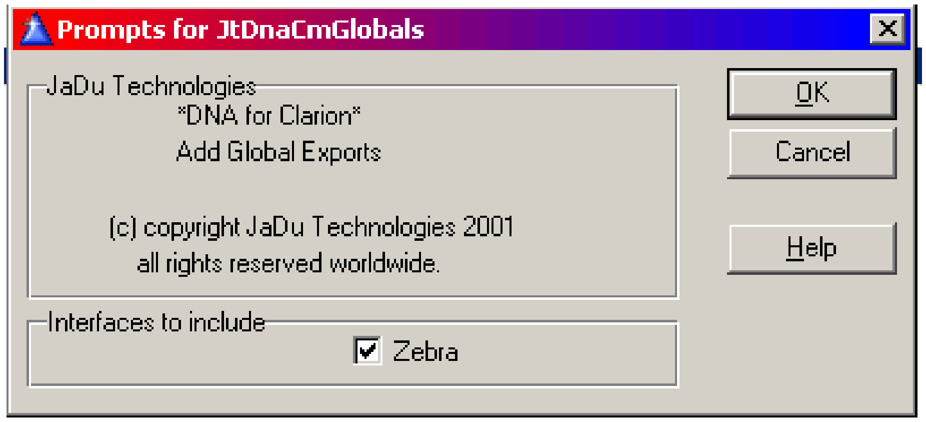
```
#CODE(JtDnaCmGlobals,'(JaDuTech-CM) DNA Global Exports'),|
    DESCRIPTION('(JaDuTech-CM) DNA Global Exports')
  #PROMPT('Zebra',CHECK),%JtDnaIncludeZebra,DEFAULT(1)
#ENDBOXED
  INCLUDE('JtCmDna.inc'),ONCE
#IF(%JtDnaIncludeZebra)
 INCLUDE('JtCmDnaZebra.inc'),ONCE
#ENDIF
```

The Class template

The class template is placed in the Local Data embed of the procedure you want to add Zebra stripes too. It instantiates the selected class and allows the programmer to give the object a unique label. Once again, it only contains an option for the Zebra class but could be easily extended to cover others.

```
#CODE(JtCmDnaClass,'(JaDuTech-CM) DNA Class Selection'),|
       DESCRIPTION('(JaDuTech-CM) DNA Class Selection')
#PROMPT('Interface Type',OPTION),%JtDnaInterface,DEFAULT('Zebra')
#PROMPT('Zebra',RADIO)
#PROMPT('Object Name:',@S255),%JtDnaClass|
    ,DEFAULT('JtDna' & %ActiveTemplateInstance)
```

Figure 1 shows the code template window.


Prompts for JtDnaCmGlobals

JaDu Technologies
"DNA for Clarion"
Add Global Exports

(c) copyright JaDu Technologies 2001
all rights reserved worldwide.

Interfaces to include
[✓] Zebra

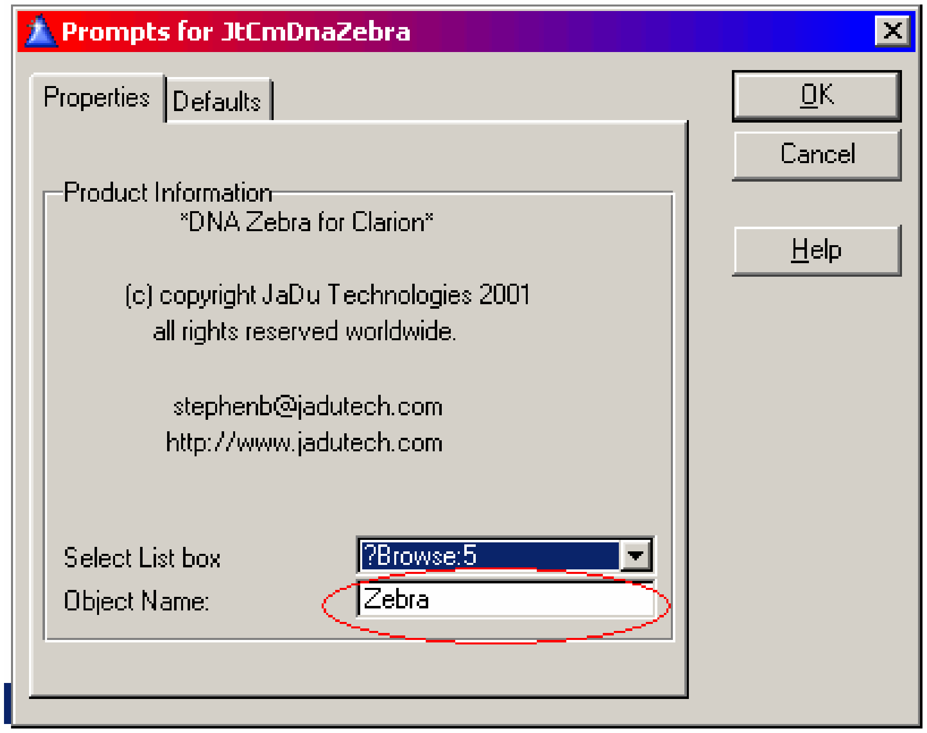
Figure 1. The code template window

The initialization template

The initialization template is placed in an embed after the window has been opened. It allows the programmer to select which list box will be cloned, which instance of the class (the label supplied in the Class Template) to use and the colour of each of the stripes. This template simply calls the Init method and loops through the list box fields looking for those that have the COLOR attribute set. For each one, call the AddStrip method with the selected colour information.

```
#CODE(JtCmDnaZebra,'(JaDuTech-CM) Browsebox Zebra'),|
   DESCRIPTION('(JaDuTech-CM) Browsebox Zebra'),REQ(JtCmDnaClass)
#ATSTART
  #DECLARE(%JtDnaValueConstruct)
  #DECLARE(%JtDnaCounter)
  #DECLARE(%JtDnaCol)
#ENDAT
#SHEET
  #TAB('Properties')
    #DISPLAY ('')
      #PROMPT('Select List box',CONTROL),%JtDnaControl,REQ
      #PREPARE
        #FIX(%Control,%JtDnaControl)
      #ENDPREPARE
      #VALIDATE(%ControlType = 'LIST','Must select a list control')
      #PROMPT('Object Name:',@S255),%JtDnaClass,DEFAULT('JtDna' |
        & %ActiveTemplateInstance)
    #ENDBOXED
    #DISPLAY ('')
  #ENDTAB
  #TAB('Defaults')
    #BOXED('Set Normal colours')
      #PROMPT('&Foreground Normal:',COLOR),|
         %JtDnaNormalForegroundNormal,DEFAULT(-1)
      #PROMPT('&Background Normal:',COLOR),|
         %JtDnaNormalBackgroundNormal,DEFAULT(-1)
      #PROMPT('&Foreground Selected:',COLOR),|
         %JtDnaNormalForegroundSelected,DEFAULT(-1)
      #PROMPT('&Background Selected:',COLOR),|
         %JtDnaNormalBackgroundSelected,DEFAULT(-1)
    #ENDBOXED
    #DISPLAY('')
    #BOXED('Set Zebra colours')
      #PROMPT('&Foreground Normal:',COLOR),|
         %JtDnaZebraForegroundNormal,DEFAULT(-1)
      #PROMPT('&Background Normal:',COLOR),|
         %JtDnaZebraBackgroundNormal,DEFAULT(-1)
      #PROMPT('&Foreground Selected:',COLOR),|
         %JtDnaZebraForegroundSelected,DEFAULT(-1)
      #PROMPT('&Background Selected:',COLOR),|
         %JtDnaZebraBackgroundSelected,DEFAULT(-1)
    #ENDBOXED
  #ENDTAB
#ENDSHEET
#FIX(%Control,%JtDnaControl)
%JtDnaClass.Init(%Window,%Control,%ControlFrom)
#SET(%JtDnaCounter,0)
#FIX(%Control,%JtDnaControl)
```

```
#IF(%Control NOT=%JtDnaControl)
  #ERROR('List box control not found!')
  #ABORT
#ENDIF
#FOR(%ControlField)
  #SET(%JtDnaCounter,%JtDnaCounter + 1)
  #SET(%JtDnaValueConstruct,%ControlField)
  #IF(%ControlFieldHasColor)
        #SET(%JtDnaCol,%JtDnaCounter + 1)
%JtDnaClass.AddStripe(%JtDnaCol,%JtDnaNormalForegroundNormal,|
  %JtDnaZebraForegroundNormal)
        #SET(%JtDnaCol,%JtDnaCounter + 2)
%JtDnaClass.AddStripe(%JtDnaCol,%JtDnaNormalBackgroundNormal,|
  %JtDnaZebraBackgroundNormal)
        #SET(%JtDnaCol,%JtDnaCounter + 3)
%JtDnaClass.AddStripe(%JtDnaCol,%JtDnaNormalForegroundSelected,|
  %JtDnaZebraForegroundSelected)
        #SET(%JtDnaCol,%JtDnaCounter + 4)
%JtDnaClass.AddStripe(%JtDnaCol,%JtDnaNormalBackgroundSelected,|
  %JtDnaZebraBackgroundSelected)
    #SET(%JtDnaCounter,%JtDnaCounter + 4)
  #ENDIF
  #IF(%ControlFieldHasStyle)
    #SET(%JtDnaCounter,%JtDnaCounter + 1)
  #ENDIF
  #IF(%ControlFieldHasIcon)
    #SET(%JtDnaCounter,%JtDnaCounter + 1)
  #ENDIF
  #IF(%ControlFieldHasTree)
    #SET(%JtDnaCounter,%JtDnaCounter + 1)
  #ENDIF
#ENDFOR
```



Product Information
DNA Zebra for Clarion

(c) copyright JaDu Technologies 2001
all rights reserved worldwide.

stephenb@jadutech.com
http://www.jadutech.com

Select List box: ?Browse:5
Object Name: Zebra

Figure 2. The initialization template window.

Of course the templates could be combined into one or two for your favorite browse template.

There you have it, two classes, three templates and from now on, three minutes to genetically modifying any queue loaded list box into a Zebra.

How easy is it?

Here's how you can hand code a list box using the Zebra class.

```
  PROGRAM
  !Make sure we include the class source
  INCLUDE('JtCmDnaZebra.inc'),ONCE
  MAP
Main    PROCEDURE
  END
  CODE
  Main
Main    PROCEDURE
Counter SHORT,AUTO
!Create an instance of the Zebra class
Zebra   JtCmDnaZebraClass
ZebraQ  QUEUE
Id        STRING(10)
nFg       LONG
nBg       LONG
sFg       LONG
sBg       LONG
        END
Window WINDOW('Caption'),AT(,,111,100),SYSTEM,GRAY,RESIZE
       LIST,AT(0,0),USE(?List),FULL,VSCROLL,|
         FORMAT('20L(2)|M*'),FROM(ZebraQ)
     END
CODE
 open(Window)
 !Initialise the Zebra
 Zebra.Init(Window,?List,ZebraQ)
 !Add stripe infdormation for each colour entry
 Zebra.AddStripe(2,-1,-1)
 Zebra.AddStripe(3,-1,16776960)
 Zebra.AddStripe(4,-1,-1)
 Zebra.AddStripe(5,-1,16744448)
 Accept
   case event()
   of EVENT:OpenWindow
     setcursor(CURSOR:Wait)
     loop Counter = 1 to 100
       ZebraQ.Id = 'Record ' & Counter
       ZebraQ.nFg = COLOR:None
       ZebraQ.nBg = COLOR:None
       ZebraQ.sFg = COLOR:None
       ZebraQ.sBg = COLOR:None
       add(ZebraQ)
```

```
      end
      setcursor
    end
  end
  close(Window)
```

Figure 3 shows the list box in action.

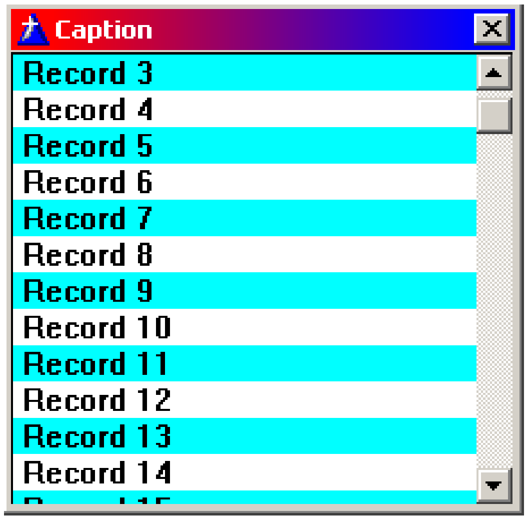


Figure 3. The Zebra list box

I've covered the basics of using a VLBProc. What has been created here could obviously be extended to include user defined runtime colours and toggling the effect on/off. How about getting cells to flash on under certain conditions? I leave it to your imagination to take over from here.

Download the source

Steve Bottomley is a long time user of Clarion, a member of Team Topspeed, and works for the Australian Government.

Reader Comments

Add a comment

Clarion Magazine

The Clarion Advisor: Displaying Clarion Dates In Excel

by Jeff Slarve

Published 2002-12-20

If you have a CSV file or other data that contains an unformatted Clarion date, and you wish to view the date in Excel, you'll need to convert the Clarion standard date to an Excel date.

Fortunately, that's easy to do. Just create a formula in another column, subtracting 36161 from the value and formatting it as a date. Clarion standard dates and the Excel DATE() function both have an arbitrary "Day 1" - for Excel, this is 1/1/1900, while Clarion's Day 1 is 12/28/1800 (although the first usable date is Day 4, or 1/1/1801). The number 36161 is the difference in days between these two dates.

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

Reader Comments

Add a comment

Jeff, Thanks, great tip. Actually, since I have to...

Clarion Magazine

Web Validation From Your Clarion App Using NetTalk

by Mark Riffey

Published 2002-12-20

Recently, I had a need for one of my Clarion programs to access a SQL database hosted on the web in order to determine if the customer's access to a service had expired. It was a fairly simple task using CapeSoft NetTalk, so I'll demonstrate how I did it.

I've made up a scenario where the Clarion program is started with the customer's customer number as a command line parameter. The customer number is passed to a web page as a parameter. The parameter is used to query a database and return an expiration date for the service that this program provides. The Clarion application parses this page to extract the necessary information.

So, how do I access my web-based database?

If the program was for in-house use and needed to access a web-hosted SQL database, I would have the program access the database directly by coding the server, database, username and password on the owner parameter of the appropriate file in the DCT.

The gotcha when using direct SQL access is that you must open port 1433 (the default MS SQL port for remote access). Since this is an application that a customer would use, I don't want to introduce any firewall issues, which will cause unnecessary technical support, which is something none of us usually wants. As a result, I'll use port 80 (the default browser port) and a web validation page with the Clarion program so that potential firewall issues are eliminated.

The validation process requires that the Clarion program confirm the service expiration date. Among other things, the expiration date is kept on the hosted SQL server database which is automatically updated by an in-house customer database maintenance app.

In my scenario, the program in question is used to provide a monthly service. The monthly subscription service is a low cost item that shouldn't require intervention and is ideally automated by a system such as this.

First, the Clarion program uses `COMMAND()` to get the customer number from the command line. It uses that parameter to complete the URL to retrieve, which in this case is:

http://www.granitebear.com/cmagvalidate.asp?custno=<somevalue>

(where "somevalue" is the customer number).

In order to retrieve the page, the program will "act like a browser" using NetTalk's WebClient object. I won't go into the specifics of how WebClient works, since that documentation is available with the product. The following code tells the NetTalk client to go to the site/page in question:

```
setcursor (CURSOR:Wait)
loc:webdata = '' ! Clear the text on the screen
! You start by telling the object what to download.
! You can choose to either download the whole page or just
! the web page header.
ThisWebClient.SetAllHeadersDefault()
ThisWebClient.CanUseProxy = 1
ThisWebClient.HeaderOnly = 0  ! We want the whole page
ThisWebClient.Fetch('www.granitebear.com/cmagvalidate.asp?custno=' |
  & loc:custno )
if ThisWebClient.Error
  Message ('This WebSite could not be downloaded. Error ' |
    & ThisWebClient.Error & ' = ' & ThisWebClient.InterpretError())
  setcursor
end
```

The Clarion program will send an HTTP GET request to that address, retrieve the resulting page, and parse the page. The page returns the service expiration date in the format `VALIDATE=date`which the program can then process.

When the page is returned, NetTalk posts the `PageReceived` event. Here's the code I used to parse the page. Most of the code is used to figure out what the ASP date is. ASP date formatting isn't as easy as `format(datefield,@d02)`, unfortunately.

```
if ThisWebClient.PageLen <= 0
 loc:webdata = ''
elsif ThisWebClient.PageLen < 30000
 loc:webdata = ThisWebClient.Page [1 : ThisWebClient.PageLen]
else
 loc:webdata = |
   '<<13,10,13,10>' |
   & ThisWebClient.Page[ThisWebClient.PageLen - 28000|
      : ThisWebClient.PageLen]
  end!if
  setcursor
  if instring('INTERNAL SERVER ERROR',loc:webdata,1,1)
    loc:webdate = 0
  else
    i# = instring('VALIDATE=',loc:webdata,1,1)
  end!If
  !format is VALIDATE=12/22/2002
  ! but asp is lame at formatting dates so
  ! we have to check each of 4 formats
  if loc:webdata[i#+10] = '/' and loc:webdata[i#+13] = '/'
    ! 1/22/3333
    ! 0111111111
    ! 9012345678
    loc:webdate = date(loc:webdata[i#+9],loc:webdata[i#+11:i#+12]|
      ,loc:webdata[i#+14:i#+17])
  elsif loc:webdata[i#+10] = '/' and loc:webdata[i#+12] = '/'
    ! 1/2/3333
    ! 0111111111
    ! 9012345678
    loc:webdate = date(loc:webdata[i#+9],loc:webdata[i#+11],|
      loc:webdata[i#+13:i#+16])
  elsif loc:webdata[i#+11] = '/' and loc:webdata[i#+14] = '/'
    ! 11/22/3333
    ! 01111111111
    ! 90123456789
    loc:webdate = date(loc:webdata[i#+9:i#+10],loc:webdata[i#+12:i#+13]|
      ,loc:webdata[i#+15:i#+18])
  elsif loc:webdata[i#+11] = '/' and loc:webdata[i#+13] = '/'
    ! 11/2/3333
    ! 0111111111
    ! 9012345678
    loc:webdate = date(loc:webdata[i#+9:i#+10],loc:webdata[i#+12],|
      loc:webdata[i#+14:i#+17])
  end!if
  if loc:webdate > 0
    if loc:webdate < today()
      glo:onlinestatus = 1   ! failed
      setcursor()
      Message ('Your subscription ended ' & format(loc:webdate,@d17),|
            'Cmag Web Validate',icon:asterisk)
    elsif loc:webdate = today()
      setcursor()
      Message ('Your subscription ends today.',|
            'Cmag Web Validate',icon:asterisk)
      glo:onlinestatus = 2 ! ok
    elsif loc:webdate > today() and loc:webdate < today() + 10
      setcursor()
      Message ('Your subscription ends on ' & format(loc:webdate,@d17),|
            'Cmag Web Validate',icon:asterisk)
      glo:onlinestatus = 2 ! ok
    elsif loc:webdate > today()
       glo:onlinestatus = 2 !ok
    end!if
  else
  setcursor()
  message ('A customer number was not included when starting ' |
    & 'this program. Contact someone.','Cmag Web Validate',icon:hand)
end!if
```

The example program displays the HTML in a text field solely for the purposes of this article, but normally you wouldn't have to do that. I would also use `post(event:accepted,?button1)` to automate the "click" of the button that starts the validation process when this code went to production. I didn't do this in the example code so that you could see the process work step by step.

What about the web side of things?

On the web side, the primary task I have to perform is retrieving the customer number from the URL. Once I've done that, I can use it to query my database (or whatever validation I am required to do).

To retrieve the URL parameter in ASP, I used `Request.QueryString("custno")`. `Request` is an object representing the incoming request from the user's browser. `QueryString("custno")` simply tells the ASP interpreter that I want to know the value of the `custno` parameter from the original URL. The incoming URL looks like this:

www.granitebear.com/cmvalidate.asp?custno=somevalue

so the result returned by `QueryString("custno")` is "somevalue" (without the quotes).

In the example code, I just used a simple if structure to allow a return of different values for testing purposes, but it is just as easy to execute a database query using that value.

Now I can query my online database from my Clarion program *and* avoid firewall issues as well. I used the easy-to-use NetTalk, but you could do this almost as easily with a few API calls to Wininet.dll. Ron Schofield has some articles on this on his openclarion.org web site.

Download the source

Mark Riffey has worked in the software industry, primarily in development and technical support for two internationally known enterprise software vendors, the world's premier information systems services company, a Fortune 100 manufacturer, and now Granite Bear Development. His business philosophy is simple: Be fair to your customers and yourself, surround yourself with brilliant people, work hard, be a good listener and have a little fun. Mark and his wife Jacki have two boys, Alex and Jonathan. Mark's other interests include Boy/Cub Scouting, backpacking/hiking and almost anything else outdoors, classic blues guitar, golf and photography.

Reader Comments

Add a comment

Clarion Magazine

CLASSy ASCII File Importing

by David Harms and Steven Parker

Published 2002-12-20

Class by Harms

Analysis by the other guy

In Parsing Strings In ASCII Files, I (Steve) showed how to import ASCII files into a database file without using a file declaration. Konrad Byers' ASCII file classes, a `Group` declared `Over` a string and a few assignments did all the work.

I also showed how to use the `SUB` function or string slicing to parse incoming data into units suitable for assignment to a target file's fields. One method takes time to allocate memory, the others use processing cycles. Your choice.

While any of these techniques work and are completely reliable, they are not especially dynamic. These techniques are tried and true and classic (see "Import" in the demo app). But if you find you need to handle another ASCII file, you need to create a second procedure to import it. (Or, copy and modify an existing procedure.)

A Parsing Class

After Dave saw my first article, he suggested a class to handle parsing the ASCII records. That way all the developer needs to do is pass the required data to the class and make field assignments as necessary.

What information is required?

First, the class must know each field's `Label`. A `Label` provides a way to refer to a variable.

Second, the class must know the field's length. Given the field length, its beginning and ending points in the ASCII string can be computed.

Lastly, the class must know the maximum length of the incoming string. Without this tidbit of information, it would be possible to (accidentally) use a subscript exceeding the length of the incoming data string. Very bad things would happen because string slicing does not do bounds checking and the compiler returns no warnings (much less errors).

Oh, one more thing, the class needs a way to store all this information.

Here's the declaration for `cciFixedRecordClass`, a small class that handles these requirements:

```
!ABCIncludeFile

OMIT('_EndOfInclude_',_cciFixedRecordPresent_)
_cciFixedRecordPresent_ EQUATE(1)

FixedRecordQueue      queue,type
FieldName               String(128)
StartPos                long
EndPos                  long
                      end


cciFixedRecordClass CLASS,TYPE,MODULE('ccifxdrc.clw'),|
   LINK('ccifxdrc.clw',_ABCLinkMode_),DLL(_ABCDllMode_)
FixedRecordQ          &FixedRecordQueue
Record                &String
RecordLength          long(0)
AddField              procedure(String fieldName,long len)
Construct             procedure
Destruct              procedure
GetFieldValue         procedure(String fieldName),String
SetRecord             procedure(*String rec)
SetRecordLength       procedure(long len)
                    end

_EndOfInclude_
```

AddField

The `AddField` method takes two parameters: a field `Label` and the field's length. This method has to be called for each field in the incoming file and the fields must be called in the order in which they occur in the ASCII file. `AddField` tells the class which `Labels` to use and the lengths of each field. It also computes the beginning and points of the field within the string:

```
cciFixedRecordClass.AddField  procedure(String fieldName,long len)
```

```
   code
   if ~records(self.FixedRecordQ)
      self.FixedRecordQ.StartPos = 1
   else
      get(self.FixedRecordQ,records(self.FixedRecordQ))
      self.FixedRecordQ.StartPos = self.FixedRecordQ.EndPos + 1
   end
   self.FixedRecordQ.EndPos = self.FixedRecordQ.StartPos + (len -1)
   self.FixedRecordQ.FieldName = FieldName
   add(self.FixedRecordQ)
```

The first time `AddField` is called, it is being called to handle the first field. So, the field's starting position must be one (1). Thereafter, the starting position must be the previous field's ending position plus one (1). (Now you see why calling order is important.)

The ending position is the current starting position plus the length parameter minus one (this correctly counts the initial position).

SetRecordLength

The `SetRecordLength` method takes a numeric parameter and can be called before or after `AddField`. It simply sets a property holding the maximum length of the ASCII record – the class uses this to do the bounds checking string slicing does not do. See the demo app for the lazy programmer's way to compute the length.

In the main processing loop, `SetRecord` ensures that a variable in the class contains the current ASCII record. The current ASCII record is the parameter for `SetRecord`.

GetFieldValue

Finally, `GetFieldValue` uses the field `Label` as its parameter to retrieve the information about the field. It gets the beginning and ending positions, checking that the data requested is within the length of the ASCII record:

```
cciFixedRecordClass.GetFieldValue   procedure(String fieldName)
   code
   self.FixedRecordQ.FieldName = FieldName
   get(self.FixedRecordQ,self.FixedRecordQ.FieldName)
   if errorCode()
?      message('Field ' & clip(FieldName) & ' not found in file definition')
      return ''
   end
   if self.FixedRecordQ.EndPos > self.RecordLength
      message('Subscript out of range')
      return ''
   end
   return(self.Record[self.FixedRecordQ.StartPos : self.FixedRecordQ.EndPos])
```

`GetFieldValue` then uses standard string slicing to return the field's data. The returned data is ready for assignment.

Final question: "Where is all this information held?" In a `queue.` But because a `queue` cannot be declared in a class, it is declared outside the class and a reference to a queue of that type is declared in the class. The constructor creates the queue:

```
cciFixedRecordClass.Construct              procedure
   code
   self.FixedRecordQ &= new(FixedRecordQueue)
```

Similarly, the destructor frees and cleans up the queue:

```
cciFixedRecordClass.Destruct              procedure
   code
   free(self.FixedRecordQ)
   dispose(self.FixedRecordQ)
```

Because code in `Construct`ors and `Destruct`ors is executed automatically, creating and destroying the `queue` in this way requires no action by the developer.

See the **Import – cci** menu option in the demo app to see this in action.

Going Dynamic

So far, `cciFixedRecordClass` moves parsing of the incoming string into a class and out of the main processing but is no more dynamic than the classic techniques.

Compare the processing code using a `Group, Over` (and this is only slightly different from the code using `Sub` or string slicing):

```
Loop While ~InputFile.Read(AsciiText)
  CUS:Number = INC:EmployeeID
  CUS:Badge  = INC:BadgeNumber
  CUS:LastName = INC:LastName
  CUS:FirstName = INC:FirstName
  CUS:Balance = INC:Balance
  If Access:Customer.TryFetch(CUS:CusNumkey)
    Access:Customer.Insert
  Else
    CUS:Badge  = INC:BadgeNumber
    CUS:LastName = INC:LastName
    CUS:FirstName = INC:FirstName
    CUS:Balance = INC:Balance
    Access:Customer.Update
  End
End
```

with the processing loop using the `cciFixedRecordClass`:

```
Loop While ~InputFile.Read(AsciiText)
  RecordClass.SetRecord(Asciitext)
  CUS:Number = RecordClass.GetFieldValue('EmployeeID')
  CUS:Badge  = RecordClass.GetFieldValue('BadgeNumber')
  CUS:LastName = RecordClass.GetFieldValue('LastName')
  CUS:FirstName = RecordClass.GetFieldValue('FirstName')
  CUS:Balance = RecordClass.GetFieldValue('Balance')
  If Access:Customer.TryFetch(CUS:CusNumkey)
    Access:Customer.Insert
  Else
    CUS:Badge  = RecordClass.GetFieldValue('BadgeNumber')
    CUS:LastName = RecordClass.GetFieldValue('LastName')
    CUS:FirstName = RecordClass.GetFieldValue('FirstName')
    CUS:Balance = RecordClass.GetFieldValue('Balance')
    Access:Customer.Update
  End
End
```

and you will note that these two codelets not only don’t appear very different; there is no striking difference in logic. In fact, the OOP version is a *lot* more typing.

By the way, if you’re uncomfortable with the two blocks of field assignment code being inline in your code, you can move them to a routine or local method and insert the appropriate call:

```
Loop While ~InputFile.Read(AsciiText)
  RecordClass.SetRecord(Asciitext)
  CUS:Number = RecordClass.GetFieldValue('EmployeeID')
  If Access:Customer.TryFetch(CUS:CusNumkey)
    Do LoadRecord
    Access:Customer.Insert
  Else
    Do Loadrecord
    Access:Customer.Update
  End
End
```

However, to handle a second ASCII file, a second procedure is still necessary.

If you’ve created a vertical market application, your end users may have a dozen different ASCII file layouts that they need to import into the application’s customer file. In this case, a dozen different import procedures are necessary. It’s far better to create a configurable import capability.

One way of doing this is to create a file in the dictionary that contains the fields, in the order in which they appear in the ASCII file and their lengths. The layout for this file might look like this:

```
FileSpec FILE,DRIVER('TOPSPEED'),PRE(FIL),CREATE,BINDABLE,THREAD
FileSpecKey KEY(FIL:CustomerID,FIL:Sequence),NOCASE,OPT
Record      RECORD,PRE()
CustomerID    LONG        !serial number
Sequence      LONG        !field order
FieldName     STRING(40)
FieldLength   LONG
            END
          END
```

The `CustomerID` field might use the software serial number. This would allow a single file to be distributed and relieve the developer from having to maintain a copy of this file for each end user.

Then, the following code would take care of ensuring that the `cciFixedRecordClass` gets the information it needs to parse the file:

```
L = 0            !initialize length counter
FIL:Sequence = 0    !clear key node
FIL:CustomerID = LOC:CustomerID !set key
Set(FIL:FileSpecKey, FIL:FileSpecKey)
Loop
  Next(FileSpec)
  If ErrorCode() or FIL:CustomerID <> LOC:CustomerID
    Break
  End
  L += FIL:FieldLength
  RecordClass.AddField(FIL:FieldName,FIL:FieldLength)
End
RecordClass.SetRecordLength(L)
```

In the main processing loop, where the assignments are made, all possible (allowed) fields are assigned. If a particular import specification does not contain a particular field, nothing happens, no assignment is made. Note that the class code includes a debug mode line for use during testing if a particular field does not exist in the procedure.

Because the end user never sees this file and you have complete control over the field names, it is all perfectly safe. Your customer may give you "Social Security Number" but it’s still "EmployeeID" to you, the dictionary file and the class methods. And, it is also perfectly transparent.

See "Dynamic Import" in the demo app to see this in action. Two different import files, with two different layout are provided.

Summary

`cciFixedRecordClass` may not be very large. But it is very convenient. It eliminates errors due to typos when creating import procedures (like a template, the code is pretested –

we think). And, with a little imagination, you can use it to handle a wide variety of fixed-field-length imports into a given file.

Download the source

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

Steve Parker started his professional life as a Philosopher but now tries to imitate a Clarion developer. He has been attempting to subdue Clarion since 2007 (DOS, that is). He reports that, so far, Clarion is winning. Steve has been writing about Clarion since 1993.

Reader Comments

Add a comment

Sweet work, guys. Very handy.