Clarion Magazine

INVEST in your own abilities — Clarion magazine

Free Resource Compiler Updated

An updated version of Larry Sand's resource compiler is now available on the downloads page.
Posted Friday, August 29, 2003

Clarion Magazine Summer Schedule

Clarion Magazine is on its summer publication schedule, with the usual four issues per month spread over July and August. The office was closed August 14-26 but we're back and gearing up for the fall. The next issue is scheduled for the first week of September.
Dave Harms, Publisher
Posted Thursday, August 28, 2003

Clarion 6 Is Almost Here

A candidate build for the final release of Clarion 6 has been made available to Early Access program participants.
Posted Tuesday, August 26, 2003

Data Structures and Algorithms Part XXII - Critical Path Analysis

Alison Neal's algorithms series resumes this week in Clarion Magazine, with a discussion of critical path analysis software.
Posted Wednesday, August 13, 2003

The Truth Is Out There (Sort Of)

Clarion programmers often use the standard TRUE and FALSE equates. But are these really the cornerstones of truth and falsehood? Veronica Chapman points out that not everything is what it

Articles: XML

News: XML

Clarion Magazine **subscriptions** (online only) are $49 for six months, $95 for one year and $170 for two years.

Subscribe **Renew**

SoftVelocity

News

PNet6 Available for C6 CR-1

Clarion 5.5 Essentials Training In Cape Town

SetupBuilder 5 Beta Program Full

CPCS C6 Beta For C6 CR-1

Duke Application Shell Summer Sale Ends

SURVEY

How many items, on average, are on your Windows task bar?

< 5

🟥 20.8%

6-10

🟩 50.7%

11-20

🟦 20.1%

21-30

▮ 6.9%

> 30

| 1.4%

144 responses

Previous Surveys

One Year Ago In CM

OLE Drag & Drop

The ClarionMag Limerick Contest

How To Ignore A Template: The Browse Reconsidered

seems.
Posted Wednesday, August 13, 2003

Translation, Clarion Style

Ever wonder how multi-language support really works in Clarion? In part 2 of this series, Nardus Swanevelder shows how to translate standard Clarion messages.
Posted Wednesday, August 13, 2003

Weekly PDF For August 3-9, 2003

All articles for August 3-9, 2003 in PDF format.
Posted Monday, August 11, 2003

Opinion: A Solution To The Scripting Problem?

The Clarion EVALUATE function is a great way to implement a level of scripting into your Clarion applications. But if you work EVALUATE too hard, your app may slow unacceptably. David Podger proposes a novel solution to the problem.
Posted Thursday, August 07, 2003

Translation, Clarion Style

Ever wonder how multi-language support really works in Clarion? In part 1 of this series, Nardus Swanevelder shows how to add basic translation capabilities to you applications.
Posted Thursday, August 07, 2003

A C6 Tagging Class Template Wrapper

Steve Parker recently published an article on an ABC compliant class for tagging on browses. When someone requested a template-enabled version, Jim Katz volunteered his services, and along the way discovered that creating a C6 ABC class template wrapper is easier than it used to be. In this second of two parts Jim concludes the template code.
Posted Thursday, August 07, 2003

PDF For July, 2003

08/31/2003

GTL Batch Compiler Updated

solid software Merchandise

ClarioNET Server Deployment Manager Maintenance Release

ProcedureNotes Template Updated

Ingasoftplus Holiday Schedule

Free Template Utility

Freeware Template For Multiprocessor Systems

Programming Objects In Clarion Table Of Contents

XLIB For C5/C5.5/C6 Available

RPM6(ABC) CR1 Available

PDF-XChange SDK Adds TWAIN Capability

ProcedureNotes Template

Clarion Jobs Page Updated August 10

ZipApp 1.1g

Clarion Third Party Profile Exchange Updated Aug 4/2003

Two Years Ago In CM

Clarion News - September 2001

Finding A File Using Template Code

In The Red

Three Years Ago In CM

Creating An MS OutLook-Style Menu In Clarion: Part 2

The Clarion Advisor: Better Debugging With DebugView

Clarion News - August 2000

Four Years Ago In CM

The Other Way To Use OLE - Part 2

Correction: Class And Wrapper For Handling Control Files

The Cranky Programmer - Install THIS!

All Clarion Magazine articles for July, 2003 in PDF format.
Posted Tuesday, August 05, 2003

We Will Miss This Fine Lady

If you attended the 1998 or 2000 ETC conferences, you probably had the privilege (and a privilege it was) of meeting Dessie White Wilson, Lee White's mother. On July 30, 2003, this wonderful lady left us. Visitation, memorial, and burial will be on Saturday, August 2.
Posted Saturday, August 02, 2003

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.

New Icetips Bio

The EIP Template 1.1 Beta

CapeSoft Source Code

CapeSoft Email Server Developer Edition

CapeSoft Progress 1.00 Beta

Multi-Proj Update

Sybase's ASA 9 Developer Edition

CPCS Recompiled with C6 EA5-1

ZipApp 1.1e Freeware

SealSoft ISP Problems

SQL Articles On Icetips

BoTpl 3.1

August Consultants Newsletter

RPM for C6 EA5.1

Mimer SQL Manager Demo

xFText 2.3 Clarion 6 Compatible

Search the news archive

Clarion Magazine

Data Structures and Algorithms Part XXII - Critical Path Analysis

by Alison Neal

Published 2003-08-13

In my last few articles I have discussed how to implement the Graph ADT and some of the algorithms associated with the Graph. In this article I'm going to discuss how to extend the Graph capabilities to perform Critical Path Analysis for Project Management applications.

Critical Path Analysis is important to project managers, particularly where they need to hire contractors to perform specific functions. The project manager needs to know when the contractor is required, so that a project is neither delayed by lack of skilled resources nor runs over budget due to having highly-paid contractors sitting idle waiting for the completion of other work.

By way of example I'll use the same task data as I described in my previous articles. The tasks on the right are dependent on the tasks on the left:

Concept Document	Client Quote
Research Requirements	Concept Document
Client Quote	Requirements Document
Requirements Document	Design Document
Develop Dictionary	Develop Windows
Develop Dictionary	Develop Reports
Design Document	Develop Dictionary
Develop Reports	Test
Develop Windows	Test

To adapt my Graph code to the Critical Path Analysis algorithm I need to make a few changes to both the Graph structure and the input file I am using. The main change to the graph structure is that the edges will contain the task name and the duration of each task (weight), rather than the information being stored against each node, as in the previous Graph examples. Figure 1 provides an illustration of the new graph structure:

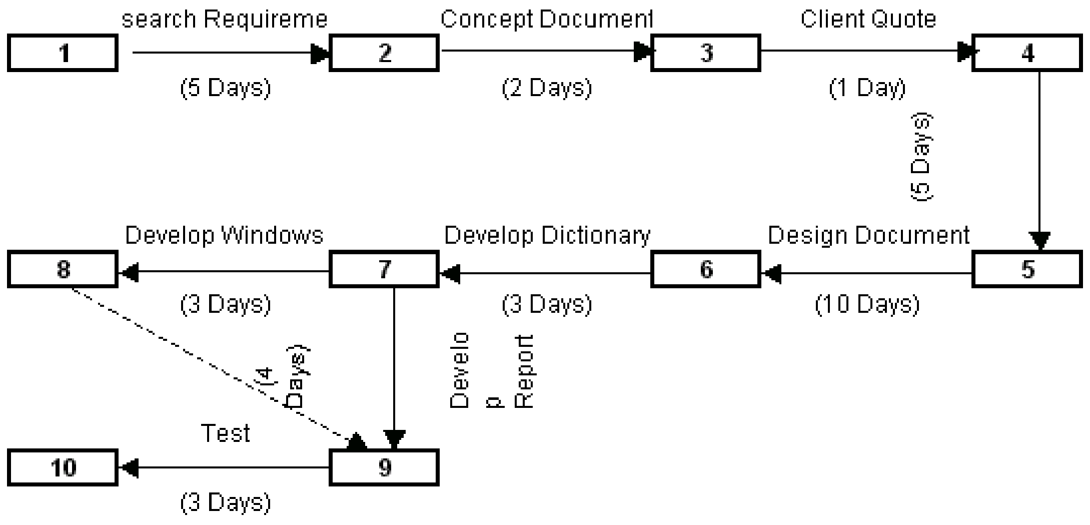


Figure 1. The requirements document

The nodes are now just sequential numbers. They could be random numbers but sequential numbers make processing a lot easier. I've had to add an extra node into the graph, which is the terminating node. Without it I would not be able to construct the final edge, as it would not have a terminating point.

Note the doted line between nodes eight and nine - this is a dummy edge as both the Windows and Report development tasks feed into testing. The edges are maintained in a matrix, or multidimensional array. The positioning is a simple process with Research Requirements that runs from node one to node two, being kept in position (1,2) of the matrix and Develop Reports taking position (7,9). So, I introduce the bogus node eight, and use the dummy edge between node eight and nine to prevent overwriting the Develop Reports edge that runs from seven to nine. If both Develop Reports and Develop Windows ran from nodes seven to nine than they would fill the same position in the matrix.

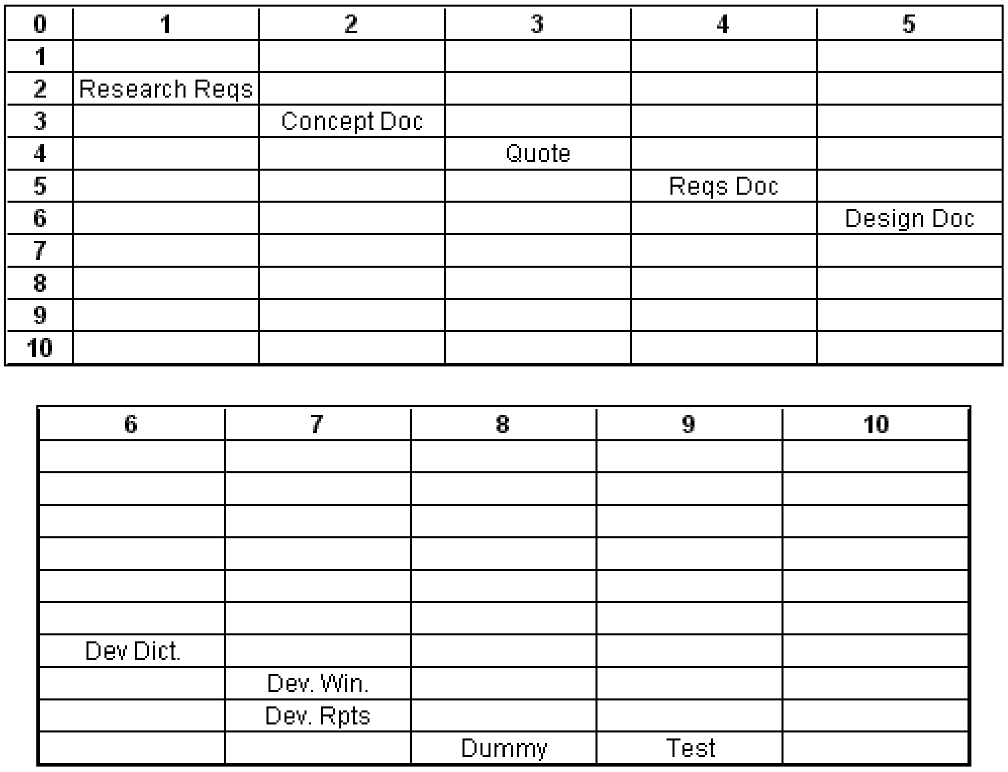
The input file then, is as follows:

```
10
1,2,Research Requirments,5
2,3,Concept Document,2
3,4,Client Quote,1
4,5,Requirements Document,5
5,6,Design Document,10
6,7,Develop Dictionary,3
7,8,Develop Windows,3
7,9,Develop Reports,4
8,9,Dummy,0
9,10,Test,3
```

The first record provides the number of nodes in the graph. For example, the second record provides the starting node (1) and the ending node (2) – these are the two nodes that the edge runs between. Then there is the edge or task name and the weight or duration of that task in days. Note the dummy edge, which has a zero weight; this corresponds to the doted line illustrated in figure 1. As it has a zero weight, the dummy edge will not affect any of the calculations used to work out the critical path.

Graph Structure

To implement the new graph structure I have used a matrix, or multidimensional array. If I were to map the example into a matrix, it would look something like Figure 2.



0	1	2	3	4	5	6	7	8	9	10
1										
2	Research Reqs									
3		Concept Doc								
4			Quote							
5				Reqs Doc						
6					Design Doc					
7						Dev Dict.				
8							Dev. Win.			
9							Dev. Rpts			
10								Dummy	Test	

Figure 2. The matrix (show in two parts)

From Figure 2 it's possible to see how the start and end nodes in the input file provide the indexing for the edge matrix. Each position in the matrix stores the address of an allocated edge node. The memory for the edge is allocated when the edge is added to the graph, and not when the matrix is dimensioned, otherwise there would be excess memory allocated for edges, which do not exist.

However, my implementation does have some overhead, as I have used strings for my array rather than simply using a multidimensional array of `LONG` data types. Unfortunately in Clarion it is not easy to maintain a dynamic array where you can increase the size of the array at run time if required. You can, however, `NEW` a larger string, copy the old data across and dispose of the old string, thus giving yourself some room to move. For the size of the graph in my example though, this is largely superfluous, as it would be quite acceptable to have a multidimensional array `DIM`med to 100 by 100 `LONG`s and still not worry about the extra memory that's not being used. Also remember that a `STRING` will use 16 bytes to store a `LONG` as each number is seen as a character, where as a `LONG` is only really four bytes in size. However, my implementation will hopefully provide you with a few more ideas on how to approach some difficult problems when you encounter them.

Here is the definition of my Graph implementation:

```
edgeNode     CLASS,TYPE
eTask        STRING(25)
eWeight      LONG
        END
GData        &STRING
EData        &EdgeNode
LongLen      BYTE(16)
NodeNum      ULONG(0)
```

The methods used are:

`Init`	Initialise the Graph
`Kill`	Clean up

Add Edge	Add the required Edge to the matrix.
CPA	Critical Path Analysis Algorithm

Because of the string allocation used, the Init method is substantially different to that used for other structures.

```
!---------------------------------------------------
graph.Init                  PROCEDURE(ULONG NumNodes)
!---------------------------------------------------
i   ULONG(0)
tmp &STRING
  CODE
  SELF.nodeNum = NumNodes
  SELF.GData &= NEW(STRING(Self.longLen * NumNodes))
  CLEAR(SELF.GData)
  tmp &= SELF.GData
  LOOP i = 1 TO numNodes
    SELF.GData &= NEW(STRING(Self.longLen * NumNodes))
    tmp[ ((i-1) * Self.longLen) + 1 : i * Self.longLen ] = ADDRESS(SELF.GData)
  END
  SELF.EData &= NULL
  SELF.GData &= tmp
```

The first thing to happen is to set the number of Nodes which are going to be held in the Graph by initialising `SELF.nodeNum` to the value passed to the function. Then the memory is allocated for one string to contain the expected number of `LONG` values - there are 10 in this case, so the string is initialised to a length of 10 * 16 positions, or 160.

Each of the positions in the string is going to contain the memory address for another `STRING` of the same size, thus providing a mechanism for the multidimensional array. So for each node 1 through 10 another string is allocated to a length of 160, and its memory address is stored in the original string. This gives one dynamically allocated multidimensional array stored in `GData`.

To add an edge, I call the `addEdge` method. Here's the source:

```
!---------------------------------------------------
graph.addEdge PROCEDURE(ULONG Node1, ULONG Node2, *STRING EdgeName, ULONG Weight)
!---------------------------------------------------
tmp1        &String

  CODE
  IF Node1 > SELF.NodeNum OR Node2 > SELF.NodeNum THEN RETURN -1.
  !Store the address of the edge in the right location of the matrix
  SELF.Edata &= NEW(EdgeNode)
  IF SELF.Edata &= NULL THEN RETURN -1.
  SELF.EData.eTask = EdgeName
  SELF.EData.eWeight = Weight

  tmp1 &= (SELF.GData[((Node1 - 1) * Self.longLen) + 1 : Node1 * Self.longLen])
  tmp1[((Node2 - 1) * Self.longLen) + 1 : Node2 * Self.longLen] = ADDRESS(SELF.EData)
  RETURN 0
```

The edge details are passed to the method, and a check is made to see whether this involves any nodes greater than those expected. I'm only expecting 10, so if the start or end node of the new edge were higher than that I can either return immediately (as above), or I could grow the size of my strings (multidimensional array) by creating new larger ones, copying the old data into them, and then disposing of the old strings. I wouldn't want to do this for every new edge over the expected number; it would probably be better is I increased the size of the strings (arrays) by a significant number, otherwise I would end up having large processing delays.

Next, the memory for the Edge is allocated, with the weight and task name being initialised to the passed values. The address of the Edge is then stored in the appropriate position of the matrix.

The `Kill` method should be reasonably straightforward now:

```
!-----------------------------------------------------
graph.Kill                  PROCEDURE()
!-----------------------------------------------------
i           ULONG(0)
j           ULONG(0)
tmp         &String
  CODE

  LOOP i = 1 TO SELF.NodeNum
    tmp &= (SELF.Gdata[ ((i - 1) * SELF.longLen) + 1 : i * SELF.longLen ])

    LOOP j = 1 TO SELF.NodeNum

      IF tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ]
        SELF.Edata &= (tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ])

        DISPOSE(SELF.Edata)
      END
    END
    DISPOSE(tmp)
  END
  DISPOSE(SELF.GData)
  RETURN 0
```

For each position in the multidimensional array, where an Edge data address has been stored, the edge needs to be disposed of, then each string whose address is stored in `Gdata` is disposed of, and last of all `Gdata` is disposed of.

Critical Path Analysis

The information I require from the Critical Path Analysis algorithm includes the duration of each task (its weight), the earliest start time (EST), the latest possible finish time (LFT) and how much spare time that task has. The eventual output of my example should be something like this:

```
Critical Path Analysis
(Tasks marked * are on the critical path)
Task                            Time    EST     LFT     Spare
Research Requirments            5       0       5       0*
Concept Document                2       5       7       0*
Client Quote                    1       7       8       0*
Requirements Document           5       8       13      0*
Design Document                 10      13      23      0*
Develop Dictionary              3       23      26      0*
Develop Windows                 3       26      30      1
Develop Reports                 4       26      30      0*
Test                            3       30      33      0*
```

Note that the Develop Windows task has one spare day, as this task is performed in parallel to developing the reports, which takes one day longer. This means that the window development could blow out by a day, without any effects on the overall project. However, it must still be finished (LFT) by the 30th day for the testing task to start on time.

Here’s the code for the CPA (Critical Path Analysis) method:

```
!----------------------------------------------------
graph.CPA   PROCEDURE()
!----------------------------------------------------
INT_MAX     EQUATE(2147483647)
i           ULONG(0)
j           ULONG(0)
k           ULONG(0)
w           ULONG(0)
tw          ULONG(0)
est         ULONG(0),DIM(SELF.NodeNum)
lft         ULONG(INT_MAX),DIM(SELF.NodeNum)
iDegree     ULONG(0),DIM(SELF.NodeNum)
oDegree     ULONG(0),DIM(SELF.NodeNum)
res         STRING(1)
!expLine     &STRING
Q           myQueue
tmp         &String
  CODE
  LOOP i = 1 TO SELF.NodeNum
    tmp &= (SELF.Gdata[ ((i - 1) * SELF.longLen) + 1 : i * SELF.longLen ])
    LOOP j = 1 TO SELF.NodeNum
      IF tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ]
        SELF.EData &= (tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ])
        IF SELF.EData.EWeight >= 0
          oDegree[i] += 1
          iDegree[j] += 1
        END
      END
    END
  END
```

Firstly the in-degree and out-degree of each node is calculated.

iDegree		1	2	3	4	5	6	7	8	9	10
		0	1	1	1	1	1	1	1	1	2

oDegree		1	2	3	4	5	6	7	8	9	10
		1	1	1	1	1	1	2	1	1	0

The first node has an in-degree of zero as there are no tasks prior to it, and the last node has no out-degree, as there is nothing that follows it. The last task, testing, does however have an in-degree of two, as two edges terminate at it, both the windows and report development. Node 7 has an out-degree of two, as both the windows and report development occur directly after it.

The starting node is then added to a queue. I’ve utilised my own queue structure, which was covered in an earlier article. The starting node is identified, as it has a zero in-degree value

```
LOOP i = 1 TO SELF.NodeNum
    IF iDegree[i] = 0
      Q.enqueue(i)
    END
  END
```

Now I want to calculate the earliest start time (EST) for each task. At this stage if the queue is empty, then there must be a cycle in the graph, that is, there is no node that does not rely on another task being finished prior to it being started. It would fundamentally mean that there was an infinite loop in the graph.

```
LOOP k = 1 TO SELF.NodeNum
    IF Q.isEmpty() THEN RETURN -1. !Graph must have a cycle

    i = Q.front()
    Q.deQueue()
    LOOP j = 1 TO SELF.NodeNum
      tmp &= (SELF.Gdata[ ((i - 1) * SELF.longLen) + 1 : i * SELF.longLen ])
      IF tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ]
        SELF.EData &= (tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ])
        IF SELF.EData.EWeight >= 0
          w = SELF.Edata.EWeight
          tw = est[i] + w
          IF tw > est[j]
            est[j] = tw
          END
          idegree[j] -= 1
          IF iDegree[j] = 0
            Q.enqueue(j)
          END
        END
      END
    END
  END
```

If the queue is not empty then I retrieve the value of the first node in the queue, and remove it from my holding queue. For that node, I examine every edge leading from it and calculate the cumulative lead times. In my example the embedded loop is only going to do anything when J equals 2. At this point the algorithm calculates the cumulative lead-time by adding the preceding nodes weights (durations) together. I also decrement the in-degree array for that node, so when a node has an in-degree of zero, it is added to the holding queue, and the tasks, which proceed from that node, have their EST calculated.

The EST array eventually looks like this:

• EST		1	2	3	4	5	6	7	8	9	10
		0	5	7	8	13	23	26	29	30	33

Using a similar technique (only backwards) I calculate the last possible finish time:

```
  LOOP i = 1 TO SELF.NodeNum
    IF oDegree[i] = 0
      lft[i] = est[i]
      Q.enqueue(i)
    END
  END
  LOOP k = 1 TO SELF.NodeNum
    j = Q.front()
    Q.deQueue()
    LOOP i = 1 TO SELF.NodeNum
      tmp &= (SELF.Gdata[ ((i - 1) * SELF.longLen) + 1 : i * SELF.longLen ])
      IF tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ]
        SELF.Edata &= (tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ] )
        IF SELF.EData.EWeight >= 0
          w = SELF.Edata.EWeight
          tw = lft[j] - w
          IF tw < lft[i] THEN lft[i] = tw.
          ODegree[i] -= 1
          IF ODegree[i] = 0 THEN Q.enQueue(i).
        END
      END
    END
  END
```

Now all that needs doing is the output, and it's reasonably easy at this stage to calculate any spare time that a task has by calculating the LFT, less the EST, less the task weight or duration.

```
  IF exp.dosopen('export.txt',1) THEN RETURN -1.

  IF exp.setpointer(0,0) THEN RETURN -1.
  SELf.expFile('Critical Path Analysis <13,10>' & |
    '(Tasks marked * are on the critical path)<13,10><13,10>' & |
    'Task<9><9><9><9>Time<9>EST<9>LFT<9>Spare<13,10><13,10>')

  LOOP i = 1 TO SELF.NodeNum
    tmp &= (SELF.Gdata[ ((i - 1) * SELF.longLen) + 1 : i * SELF.longLen ])
    LOOP j = 1 TO SELF.NodeNum
      IF tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ]
        SELF.Edata &= (tmp[ ((j - 1) * SELF.longLen) - 1 : j * SELF.LongLen ] )
        IF SELF.Edata.EWeight > 0
          w = SELF.EData.EWeight
          k = lft[j] - est[i] - w
          res = CHOOSE(k,' ', '*')
          SELF.expFile(SELF.Edata.eTask & '<9>' &w &'<9>' &est[i] & '<9>' &lft[j] |
                &'<9>' &k &res &'<13,10>')

        END
      END
    END
  END
  exp.dosclose()
  RETURN 0
```

Summary

Critical Path Analysis is a very useful tool for finding cycles or dependencies in projects, and for calculating required start dates and end dates for projects so that tasks can be effectively scheduled.

In my next article I will continue on the subject of graphs and discuss some interesting algorithms for finding the shortest path between two indirectly linked nodes. This is useful when considering travel applications, or again in a project management context for calculating the minimum completion requirements if you wanted to bring a task forward in time.

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

How is this different from the topological sort discussed...

Clarion Magazine

The Truth Is Out There (Sort Of)

by Veronica Chapman

Published 2003-08-13

Whenever you – and I – write applications in Clarion we are writing for a world that does not, *and cannot*, exist.

What I'm talking about here is "True" and "False".

Let me start at the very beginning.

Deep in the bowels of your computer there is, as I'm sure you know, a chip which, these days, is called a Microprocessor. It will almost certainly be based on Intel's 80386 architecture, which means – as far as this article is concerned – that it is 'native' to registering and processing 32-bit wide data.

Inside this microprocessor chip, buried even deeper, there is a large collection of transistors that – interconnected the way they are - constitute the Central Processing Unit (or CPU, for short).

It is the CPU's job to process Assembler Level (or Machine Language) instructions. I do not intend to explain all of them, but I would like to bring to your attention the two most fundamental 'Conditional Jumps' that the CPU interprets. These are known (to Assembler Programmers) as JZ (Jump on Zero) and JNZ (Jump on Non-Zero).

The way these operate is that the register or memory location to be conditionally tested is *fully-32-bit-wide-logically-ORed* (in one machine cycle i.e. on one of it's 1,000s of mega cycles per second) to create the condition on which the Jump (or not) will be based. In the next cycle of megas, the result of this ORing is used, by the CPU, to determine whether or not the condition for 'jumping' has been met.

The point I am making here is that, because the entire content of data is included in the Logical OR, then the result – *totally* zero or *not totally zero* – is the condition that is tested.

Consequently "Zero" in this situation means precisely *that*. Every single bit of the 32 was *re*set.

Similarly "Non-Zero", in this context, again means precisely what it says – one or more of the 32-bits *was* set.

The "Zero" and "Non-Zero" conditions are *exact opposites*, both physically and logically. Consequently assigning "True" for (say) "Non-Zero", means (by definition) that "False" must be "Zero". And consequently *strict logic* is maintained by such an assignment.

(Or vice versa. Assembler Programmers and Electronics Engineers can use either Positive or Negative Logic. Because the conditions and the logic are strict, it makes no difference).

However, when we come to Clarion, there is a slight shifting of ground. If you check out EQUATES.CLW you will find the two statement lines:

```
FALSE                    EQUATE (0)
TRUE                     EQUATE (1)
```

These statement lines provide us with the ability to code conditions using the words `True` and `False` for readability.

But there is a difference between this 'truth' and this 'falsity'. In the Machine Language context `True` and `False` are *exact opposites*. In the Clarion context `True` and `False` are *not* exact opposites. I submit that "1" is not the *exact opposite* of "0".

What does this mean? It means that, in Clarion, something can be both "NOT False" and "NOT True" *at one and the same time*. The value of "2" for example.

It means that, in Clarion, "NOT True" is not, necessarily, the same as `False`. And vice versa. It means that, when using Clarion's version of `True` and `False`, *unreal* situations are possible within an application.

"Unreal situations" is, of course, a euphemism for "bugs".

Clarion's usage of "1" and "0" *does* mean that `True` cannot be `False`, and likewise `False` cannot be `True`. And this is sufficient for writing applications.

As a competent programmer, you will doubtless reckon that "NOT True" possibly being different from `False` is a minor problem and, for the sake of program readability, I would agree with you. I reckon I am a competent programmer. Nevertheless, when using Clarion's BAND() and BOR() Functions, I have been caught out a few times on this one. Just remember this. The Clarion definitions of `True` and `False` are *illogical* - in the strictest sense of that word.

On the bright side, however, all 'higher level' languages (C, C++, Visual BASIC, etc. etc.) suffer from the same delusion, and therefore Clarion is no different – in that respect – from any them. Consequently a `True` or `False` return from a Windows API Function would be interpreted correctly by a Clarion `"IF WinAPIFunction() = True … ELSE … END"`, and so on.

However, one word of caution. Make sure that you read the documentation correctly. Do not expect a Windows API function that is defined as returning a 'non-zero' condition, to be the same condition as Clarion `True`. And furthermore 'non-zero' does not mean GREATER THAN zero. Only `"IF WinAPIFunction() NOT = 0 … ELSE … END"` constitutes the *correct* conditional test.

Veronica Chapman earned a B.Tech in Electronics & Electrical Engineering from Brunel University in 1968, and subsequently embarked on a career as a Programmer/Analyst, first writing code at machine level, and shortly thereafter working with real time systems and communications. By the mid '70s she was using languages such as COBOL and FORTRAN. Never a fan of BASIC or the C language, she discovered Clarion in the mid-90s and, ever since, has used it to create applications for the 16-bit (Win 3.x) and 32-bit Windows Platforms. An assembly code programmer from way back, Veronica discovered a cornucopia of very useful functions in the Windows API, and set about making these functions available to Clarion applications.

Reader Comments

Add a comment

It gets worse. When using a component compiled in another...
The VBX was probably written by a C programmer where 0...
Normal VB True = -1 and False = 0. COM calls return an...
You also must watch for string values. With Clarion...
The premise posed in this article is why it makes sense to...
The premise posed in this article is why it makes sense to...
For this reason that I Code my TRUE and FALSE...
Interesting. From a strict logic viewpoint, I would have...
The comments regarding problems with VB and VBX are, of...

Clarion Magazine

Reborn Free CLARION online

Clarion News

PNet6 Available for C6 CR-1

PNet6 for Clarion 6 CR-1 is now available for download.
Posted Thursday, August 28, 2003

Clarion 5.5 Essentials Training In Cape Town

A Clarion Essentials class will be held in Cape Town, South Africa, from October 6-9, 2003, at Die Herberg. The course will cover the following topics: Clarion Dictionary; Events and Properties; New Applications; Dynamic Link Libraries; Clarion Language Statements; Introduction to OOP; Source Embeds; Reports; Debugger; OLE/OCX; Win32 API. Durban area is next.
Posted Thursday, August 28, 2003

SetupBuilder 5 Beta Program Full

The SetupBuilder 5 beta program is full right now, and LinderSoft is not currently accepting any more beta testers. SetupBuilder 5 has been redesigned from the ground up to be a powerful, flexible, and intuitive RAD script-based installer tool. New features have been added and the ease of use has advanced even further, with full drag-and-drop support for installer creation. Even those who have never programmed an installation before can develop one in less than ten minutes. The SetupBuilder 5 IDE is completely written with C6 and has been compiled with C6 Release Candidate 1. SetupBuilder 5 is in its final development stage and the first beta is expected soon. The following Gold editions will be available: SetupBuilder for Clarion 6; SetupBuilder 5 Standard Edition; SetupBuilder 5 Professional Edition.
Posted Thursday, August 28, 2003

CPCS C6 Beta For C6 CR-1

Files for CPCS C6 Beta, built with Clarion6 CR-1, are available for download. Your current C6 Beta install codes are valid for this build.
Posted Thursday, August 28, 2003

Duke Application Shell Summer Sale Ends 08/31/2003

The Application Shell Summer Sale (50% off) will end at Midnight on the last
Posted Thursday, August 28, 2003

GTL Batch Compiler Updated

Steve Parker's Go To Lunch batch compiler has been updated with several bug fixes, and a

default to low priority for faster exports to TXAs.
Posted Thursday, August 28, 2003

solid software Merchandise

solid software has an online shop for various merchandise. You can buy things like t-shirts, sweaters, caps, mugs, etc. with solid software logos. All money goes to cafeshops.com.
Posted Thursday, August 28, 2003

ClarioNET Server Deployment Manager Maintenance Release

A maintentance release of the ClarioNET Server Deployment Manager is now available.
Posted Thursday, August 28, 2003

ProcedureNotes Template Updated

New features in the ProcedureNotes template include: Two tabs for notes on the procedure screen, one for notes and one for revision history; Optionally include notes in the generated CLW file as OMITTED text; More information available to be printed in the Comments Utility Template. The web site will be updated shortly to reflect changes in the template.
Posted Thursday, August 28, 2003

Ingasoftplus Holiday Schedule

The Ingasoftplus office will be closed August 15, 2003 through September 2, 2003. All products are still available for sale during this time, and support email is being checked periodically. After this holiday Ingasoftplus will release EasyExcel 3.00, CWPlus 2.00, EasyResizeAndSplit 2.00, EasyListPrint 1.05, chSTD 2.64, and the new MAV Direct ODBC. Many products will also soon be available for C6.
Posted Thursday, August 28, 2003

Free Template Utility

Danie de Beer has created a free utility that makes it easier to locate templates on your hard drive. The utility scans through all template files and sorts the code sections, i.e. by #CODE, #TEMPLATE, #CONTROL etc. You can also do a text search and tells you which TPW files are included in which template. Email danie@unispd.co.za to tell Danie you've downloaded it.
Posted Thursday, August 28, 2003

Freeware Template For Multiprocessor Systems

On 6/8/03 there was a thread called C55 app and Citrix concerning Clarion applications running on multi-CPU servers. The suggested solution was running IMAGECFG to bind the process to one processor instead of allowing the OS to divide the threads over the available CPUs. This free template does this automatically for applications compiled with versions prior to C6. Threads are bound to the first processor.
Posted Thursday, August 28, 2003

Programming Objects In Clarion Table Of Contents

A table of contents for Russ Eggen's book "Programming Objects in Clarion" is now available.
Posted Thursday, August 28, 2003

XLIB For C5/C5.5/C6 Available

A new version of XLIB with support for C6 is available for download. All these files are encrypted with their usual passwords. The xVIEW class with C6 support is available also, and is now shipped in encrypted form. Customers who have xVIEW sources will be informed about file name with new xVIEW version and its password by e-mail shortly.
Posted Thursday, August 28, 2003

RPM6(ABC) CR1 Available

RPM6 for CR-1 is now available for download. This install requires your RPM6 unlock codes.
Posted Thursday, August 28, 2003

PDF-XChange SDK Adds TWAIN Capability

PDF-XChange SDK has been updated to allow the control of almost all TWAIN scanner settings and functions (subject to the manufacturer's TWAIN UI implementation). This means you should be able to bypass popping the Manufacturers User Interface for the scanner and control entirely within your own app. Legacy and ABC example now available for download in both the licensed and evaluation versions. Please also note there is a separate Library DLL and documentation for this.
Posted Wednesday, August 13, 2003

ProcedureNotes Template

Eric Jacobowitz has released the ProcedureNotes template. This template is 1. A Global Extension template which will add the Local Extension to each procedure in your app; 2. A Local Extension template, which allows you to keep notes about each procedure, and; 3. A Utility Template which will create a .txt file detailing each procedure and the notes for each procedure. Price is $15.
Posted Wednesday, August 13, 2003

Clarion Jobs Page Updated August 10

Alfred Blaho has updated the Clarion Jobs page.
Posted Wednesday, August 13, 2003

ZipApp 1.1g

ZipApp 1.1g is now available for download. If you already have ZipApp, you can click on the version number to go to web site.
Posted Wednesday, August 13, 2003

Clarion Third Party Profile Exchange Updated Aug 4/2003

A new update for the Clarion Third Party Profile Exchange is now available. You must have Product Scope 32 PRO Version 4.5 or 4.5a to view profiles with data files (downloadable profiles).
Posted Wednesday, August 13, 2003

New Icetips Bio

This week's Icetips Bio goes to Eastern Europe, home of a well-known Clarionite - and IceTips' first bio from the Czech Republic. An artist-programmer with a passion for film that matches his passion for Clarion, he's also a proud father and husband. Packed full of

wonderful pictures (including a castle or two) his bio highlights his work and his views on life.
Posted Wednesday, August 13, 2003

The EIP Template 1.1 Beta

The EIP Template has been updated and is now released in a beta version. This release includes changes to the user interface so it is more simple and intuitive to set up, and field-specific validation on each column. To see the most major changes check out the More Field Assignments button on the Action tab for each column. The install program also supports Clarion 6. Currently there are no known issues with Clarion 6. The EIP Template manual will be updated when the beta period is over.
Posted Wednesday, August 13, 2003

CapeSoft Source Code

This page lists CapeSoft products which can be purchased with source code.
Posted Wednesday, August 13, 2003

CapeSoft Email Server Developer Edition

CapeSoft Email Server developer edition is a royalty free email server for you to distribute with your application that's almost exactly the same application as the normal email server, but it has an additional sending and receiving authentication layer. Only your applications will be able to talk to the server. The server will send outgoing email, and can collect emails. Features/benefits include: Royalty free (no per server/copy costs); Ideal for existing NetTalk users who want to send emails from their applications; Your application does not have to be online to send emails - your app just sends its emails to the Developer Edition and it will take care of the rest; You don't need to code direct (MX) email sending or dial-up scheduling into your application; The combination of your application with the Developer Edition can be used to send emails, independently of your client's ISP or in-house Email Server; Can be chained together in large companies with multiple branches to support CapeSoft Replicate. Price: $399.
Posted Wednesday, August 13, 2003

CapeSoft Progress 1.00 Beta

CapeSoft Progress can replace all the existing progress bars in seconds, and requires no hand coding. Features include: Real time previewing as you change the progress settings; Globally replace progress bars with a single style, or add custom progress bars to any window; Add as many types and styles of progress bars as you like; Built on the CapeSoft Draw engine. During the beta the price is $29 (gold price will be $39). Requires CapeSoft Draw.
Posted Wednesday, August 13, 2003

Multi-Proj Update

This update to Multi-Proj has revised documentation, and a new tutorial for doing driver substitution.
Posted Wednesday, August 13, 2003

Sybase's ASA 9 Developer Edition

Michael Gould points out that Sybase's ASA 9 developer edition is available as a free download.

Posted Wednesday, August 13, 2003

CPCS Recompiled with C6 EA5-1

All CPCS C6 Beta products have now been recompiled with EA5-1. New install files have been posted for download:
Posted Wednesday, August 13, 2003

ZipApp 1.1e Freeware

ZipApp 1.1e adds a CLW option to backup selection, and CHM to HLP backup selection. There are also other minor updates. If you have ZipApp just click on the version number to go to the web site and download the latest version.
Posted Tuesday, August 05, 2003

SealSoft ISP Problems

SealSoft's ISP is presently having some difficulties. In the meantime you can email SealSoft at ivanovsky@netman.ru.
Posted Tuesday, August 05, 2003

SQL Articles On Icetips

The SQL articles are back up on Icetips (e.g. the Better SQL series by Dan Pressnell). The articles are now in a database and there is a search/browse page that you can use. You can access a somewhat different page at http://www.icetips.com/articles.php but it does not have any search capabilities.
Posted Tuesday, August 05, 2003

BoTpl 3.1

BoTpl has been updated to Version 3.1. Bo_resc now includes adding your own customized multi-line resources. A new template, BrowseSearch, is a control template to drop on a window with an ABC Browsebox to allow searching records and memos on a file in an ABC Browsebox. It does not affect current ranges or filters, but does additional filtering in the TakeRecord embed.
Posted Tuesday, August 05, 2003

August Consultants Newsletter

The August Newsletter for computer consultants is now shipping. This is a service for consultants who wish to provide a newsletter to their end users. It is designed to be used as a marketing tool. Click on Sample Newsletter.
Posted Tuesday, August 05, 2003

RPM for C6 EA5.1

RPM is now available for C6 EA5.1.
Posted Tuesday, August 05, 2003

Mimer SQL Manager Demo

This is a demo of a product under development to manage SQL databases as part of the Riebens Business Engineering Suite. Mimer.com has a SQL 99 database that is accessible

through ODBC. Mimer has a very small footprint and can be used as an embedded SQL engine, a mobile device engine, and as a SQL server. Mimer SQL Manager from Riebens Systems, shows how Clarion can connect to the SQL engine via ODBC and manage the database. You can create databanks, schemas, tables, and columns, create tables in the same schema across different databanks, export TXDs to Mimer SQL script, view all data rows in a user created table, and create Mimer SQL objects through a script interface. Also available is an AVI file for people that do not want to download the Mimer SQL engine.
Posted Tuesday, August 05, 2003

xFText 2.3 Clarion 6 Compatible

Updated xFText demonstration program and install iits for Clarion 6(EA5), and for Clarion 5.5 & 5 are available.
Posted Tuesday, August 05, 2003

Clarion Magazine

Translation, Clarion Style

by Nardus Swanevelder

Published 2003-08-13

Last week I looked at translating the text on the application windows, but what about standard Clarion messages? As I mentioned last week, the ABUTIL.TRN file contains translation pairs for all the user interface text generated by the ABC Templates and the ABC Library, and if you are going to design all of your applications in a specific foreign language then it would be a good idea to edit the ABUTIL.TRN file directly. Remember to make a backup first. The changes to this file will be available to all newly designed apps or procedures.

Text that can be translated by changing the ABUTIL.TRN file includes 'Setup Printer', 'Stack all open windows', 'Browse' etc. If the application's language will be changed at runtime it will not be necessary to change this file, because all the text in the application will be translated at runtime; more on this in the next article.

You also have to deal with Clarion's error messages. If you use the ABC template chain the errors are handled by the **ErrorClass**, explained by Russ Eggen as:

> The purpose of the ErrorClass is to report an error condition and indicate the severity of the error.

The messages and text to report the error and severity of the error are stored in the ABERROR.TRN file.

If you use one language per EXE, you can create multiple copies of the ABERROR.TRN file and using something like Capesoft's Multiproject to compile different EXE's with different ABERROR.TRN files. Multiproject allows you to compile different EXE's with different settings and include files. If you do not have Multiproject then this becomes a tedious task because you would have to change the ABERROR.TRN file manually before compiling each language EXE.

I suggest you read Russ's article Making Sense of ABC's ErrorClass if you need to understand this class better.

Another resource is the Clarion online Help; just search for "ErrorClass Multi-Language Capability." I have had a look at the help and decided to follow its advice to overrides the `ErrorClass.Init` method by deriving my own error class. To do this I created the following include file:

```
MyError.Inc
!ABCIncludeFile
 OMIT('_EndOfInclude_',_MyErrorClassPresent_)
```

```
_MyErrorClassPresent_  EQUATE(1)
 Include('Aberror.inc')
 Include('Aberror.trn')
Language:English       EQUATE(0)
Language:Afrikaans     EQUATE(1)
MyErrorClass    CLASS(ErrorClass),TYPE, |
    MODULE('MYERROR.CLW'), LINK('MYERROR.CLW',_ABCLinkMode_),|
    DLL(_ABCDLLMode_)
MyInit   PROCEDURE(BYTE PreferredLanguage)
                      END
_EndOfInclude_
```

See my article Adding Page Of Pages To A Clarion Report for a explanation of deriving from an ABC class.

The include file as listed above derives a new class from `ErrorClass`, and create a new method called `MyInit` that tells the error class what language it should use.

```
  MEMBER
  Map
  End
  INCLUDE('MYERROR.INC'),ONCE
MyErrorClass.MyInit PROCEDURE(BYTE PreferredLanguage)
    CODE
    SELF.Errors &= NEW ErrorEntry        !allocate new Errors list
    CASE PreferredLanguage               !which language was selected
    OF Language:Afrikaans                !if Afrikaans
        SELF.AddErrors(AfrikaansErrors)  !add Afrikaans errors
    ELSE                                 !otherwise...
        SELF.AddErrors(DefaultErrors)    !add default (English)
    END
```

I can hear some of you asking but how would the app know what AfrikaansErrors are?

If you look at the MyError.Inc file again you will see that a file called ABERROR.TRN is included; by investigating this file you can answer this question. Here is a partial listing of ABError.Trn:

```
DefaultErrors GROUP
Number USHORT(44)
       USHORT(Msg:RebuildKey)
       BYTE(Level:Notify)
       PSTRING('Invalid Key')
       PSTRING('%File key file is invalid. Key must be rebuilt.')
       USHORT(Msg:RebuildFailed)
       BYTE(Level:Fatal)
       PSTRING('Key was not built')
       PSTRING('Error: (%ErrorText) repairing key for %File.')
       USHORT(Msg:CreateFailed)
    …
End
```

The ABError file by default contains a single `Group` structure called `DefaultErrors`, and all you have to do to add multi-language error messages to your app is to add a group for each language that you want to support. Here are the first couple of lines from the Afrikaans group that I added to the ABError.Trn file:

```
AfrikaansErrors GROUP
Number USHORT(44)
       USHORT(Msg:RebuildKey)
       BYTE(Level:Notify)
       PSTRING('Nie geldige Sleutel')
       PSTRING('%File sleutel leêr is ongeldig. Sleutel moet herbou word.')
       USHORT(Msg:RebuildFailed)
       BYTE(Level:Fatal)
       PSTRING('Sleutel was nie herbou nie')
       PSTRING('Fout: (%ErrorText) herstel van %File.')
    …
End
```

Note the use of the macro symbols `%ErrorText` and `%File` in the Afrikaans `Group` structure.

To implement the new error class follow these steps:

1. Copy the three files (MyError.Inc, MyError.clw and the new ABError.Trn) to the Libsrc folder.
2. To activate the new class in the app go to Global Properties, scroll to the **Classes** tab, click on **Refresh Application Builder Class** and then click on **General** and change the **Error Manager** to `MyErrorClass`. (Remember that you have to do this in each app file if you have a multi-dll app.)
3. Go to the Global Embeds, ABC Embeds, **Error Manager (MyErrorClass), Init** and add the following after the parent call:

```
INIMgr.Init('.\MyApp.INI')
GLB:Language = INIMgr.Fetch('Preferences','Language',1)
GlobalErrors.MyInit(GLB:Language)
```

The last line of code calls the new `MyError` class method `MyInit`, which tells the application which group of error messages should be included in the app. Remember that the ABError.Trn file gets compiled into the application and the exe will thus increase in size.

The ABError.TRN file does not cover all the error messages and that brings as to the usage of an Environment file.

What is an environment file? The Clarion help says:

> An environment file contains internationalization settings for an application. On program initialization, the Clarion run-time library attempts to locate an environment file with the same name and location as your application's program file (appname.ENV). If an environment file is not found, the run-time library defaults to standard English/ASCII.
>
> The .ENV file is compatible with the .INI files used by Clarion if the CLACHARSET is set to OEM, because Clarion .INI files are generally written using OEM ASCII, not the ANSI character set.

The LOCALE procedure can be used to load environment files at run-time to dynamically change the international settings. LOCALE can also be used to set individual entries.

What does an environment file look like? It is a normal text file that consists of entries and values. I have listed the file contents below with some comments describing the meaning of each entry and value.

```
!This determines the character set used by the ↵
  entries in the .ENV file
!Default setting is as follows:
!CLACHARSET=WINDOWS
!Specifies a specific collating sequence for  ↵
  use at run-time. This collating sequence is  ↵
  used for building KEY and INDEX files, as well  ↵
  as for sorting QUEUEs and all string/character comparisons.
!Default setting is as follows:
!CLACOLSEQ="AÄÅÆaàáâäåæBbCÇcçDdEÉeèéêëFfGgHhI ↵
  iìíîïJjKkLlMmNÑnñOÖoòóôöPpQqRrSsßTtUÜuùúûüVvWwXxYyZzÿ"
!This specifies the text used to indicate AM or  ↵
  PM as a part of a time display field
!Default setting is as follows:
!CLAAMPM="AM","PM"
!To change to Afrikaans use the following:
CLAAMPM="VM","NM"
!Specifies the text returned by procedures and  ↵
  picture formats involving the month full name.
!Default setting is as follows:
!CLAMONTH="January","February","March","April", ↵
  "May","June","July","August","September", ↵
  "October","November","December"
!To change to Afrikaans use the following:
CLAMONTH="Januarie","Februarie","Maart","April", ↵
  "Mei","Junie","Julie","Augustus","September", ↵
  "Oktober","November","Desember"
!Specifies the text returned by procedures and  ↵
  picture formats involving the abbreviated month name
!Default setting is as follows:
!CLAMON="Jan","Feb","Mar","Apr","May","Jun", ↵
  "Jul","Aug","Sep","Oct","Nov","Dec"
!To change to Afrikaans use the following:
CLAMON="Jan","Feb","Mar","Apr","Mei","Jun","Jul", ↵
  "Aug","Sep","Okt","Nov","Des"
!This allows Digraph characters to collate correctly.  ↵
  A Digraph is a single logical character that is  ↵
  a combination of two characters (Char1 and Char2).  ↵
  The Digraph is collated as the two characters that  ↵
  combine to create it. They are more common in  ↵
  non-English languages. For example, with CLADIGRAPH= ↵
  "ÆAe,æae" specified, the word "Jæger" sorts before  ↵
  "Jager" (since "Jae" comes before "Jag").
!CLADIGRAPH="ÆAe,æae"
!Allows you to specify upper and lower case letter pairs.  ↵
```

```
  The WINDOWS setting uses the default upper/lower case  ↵
  pair sets as defined by the Windows Country setting  ↵
  (in the Control Panel)
!CLACASE="ÄÅÆÇÉÑÖÜ","äåæçéñòü"
!Default text for buttons
!Default setting is as follows:
CLABUTTON="OK","&Yes","&No","&Abort","&Retry", ↵
  "&Ignore",Cancel","&Help"
!To change to Afrikaans use the following:
CLABUTTON="OK","&Ja","&Nee","&Stop","&Herhaal", ↵
  "&Ignoreer","Kanselleer","&Help"
!CLAMSGerrornumber="ErrorMessage" This allows run-time  ↵
  error messages to be overridden with translated strings
!Default settings are as follow:
!CLAMSG2="No File Found"
!CLAMSG3="Path Not Found"
!CLAMSG4="Too many open Files"
!CLAMSG5="Access denied"
! numerous lines omitted for brevity
!CLAMSG801="Variable Not Found"
!To change to Afrikaans use the following:
CLAMSG2="Geen Leêr Gevind"
CLAMSG3="Roete is nie gevind"
```

This is all well and good, but how do you change the environment file at runtime?

Go to the Global Embeds, ABC Embeds, **Error Manager (MyErrorClass), Init** and add the following after the parent call:

```
If GLB:Language = 1
   LOCALE('Afrikaans.ENV')
END
```

Remember that you have to ship the different environment files with your application as these are not compiled into your application.

Stop(), Halt(), and Message()

Now that the Clarion Error Messages question has been covered the next question is: what about the `Stop()`, `Halt()` and `Message()` functions?

Remember I said in the beginning that this will not be a complete work on translation and on this topic I took the easy way out by using MessageBox from CapeSoft. This product replaces the standard Clarion `Halt()`, `Stop()` and `Message()` functions and add a lot more features, one of which is multi-language support.

It is possible to replace the Clarion Message function yourself or any other Clarion function for that matter. All the Clarion Runtime Library map definitions can be found in the file BUILTINS.CLW, and if you search for message in this file you will find the following line of code:

```
MESSAGE(STRING,<STRING>,<STRING>,<STRING>,|
  UNSIGNED=0,BOOL=FALSE),UNSIGNED,PROC,|
  NAME('Cla$MessageBox')
```

You can also use the property `Prop:MessageHook` to replace this function with your own function. The Clarion Online Help has the following on the MessageHook property:

> A property of the SYSTEM built-in variable that sets the override procedure for the MESSAGE internal Clarion procedure. Equivalent to {PROP:LibHook,6}. Assign the ADDRESS of the overriding procedure, and the runtime library will call the overriding procedure instead of the MESSAGE procedure. Assign zero and the runtime library will once again call its internal procedure. The overriding procedure's prototype must be exactly the same as the MESSAGE procedure. (WRITE-ONLY)

To enable the multi-language feature in `MessageBox` add the following code to the application:

```
case Selected_Language
of 'English'
  ThisMessageBox.TranslationFile = path() & '\English.trn'
of 'Afrikaans'
  ThisMessageBox.TranslationFile = path() & '\Afrikaans.trn'
of 'Spanish'
  ThisMessageBox.TranslationFile = path() & '\Spanish.trn'
else
  if FileDialog('Select Translation File',ThisMessageBox.|
     TranslationFile,'Translation Files|*.trn|All Files|*.*',16) .
end
```

When the application is run and it calls a `MESSAGE()`, `STOP()` or a `HALT()` function, the translation file will be created (if it does not exist already) as an INI-type file. The file will contain a section for the button text and one for other miscellaneous text. Here is a sample of a typical file:

```
[MessageBox_Buttons]
Button:OK=OK
Button:Yes=&Yes
Button:No=&No
Button:Abort=&Abort
Button:Retry=&Retry
Button:Ignore=&Ignore
Button:Cancel=Cancel
Button:Help=&Help

[MessageBox_Text]
StopHeader=Stop
StopDefault=Exit?
HaltHeader=Halt
TimeOutPrompt=Time Out:
DontShowThisAgain=Don't Show This Again
LogTimer=Timer
```

All you have to do is edit this file and replace it with the correct text. For example:

```
[MessageBox_Buttons]
Button:OK=OK
Button:Yes=&Ja
Button:No=&Nee
[MessageBox_Text]
StopHeader=Stop
StopDefault=Verlaat?
```

It is that easy!

Date and time formats

One last question that needs an answer is what to do about date and time formats.

The easiest option is to use `@D17` and `@D18` for dates and `@T7` for time. When using `@D17` Clarion will use the Windows short date format and when using `@D18` Clarion will use the Windows long format. When using `@T7` Clarion will use the long time format from Windows. This way it is Windows' job to see that dates and times are properly formatted for the locale.

It should be possible for most end-users to change these Window settings themselves. Just go to Control Panel, Regional Settings and change the date and time formats as desired.

Yes, I know that in some instances this would not be the right solution; to solve this you can override the date and time pictures as well. I will show how to do this with a template, in an upcoming article. That template will also deal with currency indicators on entry controls.

Summary

In this article I have showed how to use the Clarion Translator Template prompts, environment files, a new derived error class and a couple of TRN files to translate an application into a different language. A couple of questions remained unanswered:

- What about currencies and measurement formats?
- What about dynamically created prompts, controls and windows?
- What about dynamically created reports?
- What about third party applications?
- How do I change the language at runtime because the solutions so far mostly give me an option of only one language per exe?

I will address these topics in the next article.

Download the source

Nardus Swanevelder was born and raised in South Africa. He was a networking engineer for seven years before he moved over to the commercial side of the business. Nardus has developed a Sale Cycle Management system for the Information and Communication Technology industry. Nardus has been programming in Clarion since 1989, and holds B.Com and MBA degrees.

Reader Comments

Add a comment

Clarion Magazine

Opinion: A Solution To The Scripting Problem?

by David Podger

Published 2003-08-07

There is a "scripting problem" in Clarion. Scripting allows an end user to control logical flows within a program. A "script" is interpreted at runtime and becomes part of the delivered logical solution.

In Clarion, the `Evaluate` function allows a software designer to include scripting within an application. A great variety of valid Clarion expressions may be evaluated at runtime using variables and functions that have been "bound," that is, made available to the parsing and processing capabilities of the `Evaluate` function. The result passed back by an `Evaluate` is a single value, or true/false if the expression is conditional.

The problem is speed. As more variables and functions are bound, as more expressions are processed, the slower things get, to the point of unacceptability.

Some Clarion programmers (*with first names beginning with "A"*) think scripting is obsolete. They advocate scrapping scripting within the parent program and substituting a process outside it. Such a process, written for instance in C#, can be compiled and its execution controlled by the parent via COM. This solution overcomes the speed problem but introduces another, which is the lack of a shared data structure between the two processes.

When `Evaluate` commands are executed by a Clarion program the results may immediately be stored in whatever data structure the programmer chooses. Say, for instance, a user-written script is processing a financial model. Such models have many inflows and outflows of cash and a number of time intervals (e.g. years) to process.

A simple data structure for a financial model is a two-dimensional matrix, in which the cash items are rows and the time intervals columns. An iterative procedure can be written to execute a series of `Evaluate` statements, each of which uses values from the matrix as input, and writes results back within that matrix.

It is not easy, however, for an external process to do the whole job in the above manner, that is, to receive a set of starting values, to iteratively calculate a complete matrix according to user-written formulas and to somehow pass this matrix back to the parent program. It is true that one or more records in a file can be used as a transfer vector between the two processes, but this creates speed issues of its own.

In a nutshell the scripting problem is: How to allow a user to control logical flows within a program without paying a speed penalty.

A possible solution?

Sometimes, of course, there is no speed penalty because user inputs are required throughout the execution of a model. Evaluation is still slow, but its lack of speed is not detectable. In cases like this, scripting executed within the one program is desirable, being less complicated than having to write two processes and organize their inter-communication.

So scripting itself is not the real problem, just its unacceptable speed in some circumstances. Perhaps the solution is to use scripting in some cases and speed it up in others. How could this be done? Forget about half-measures, only executable code runs fast. So, the answer is to substitute logically identical executable code for the interpreted script. Breaking this process down into steps:

1. A script must be converted into Clarion source.
2. The source must be compiled and linked.
3. The resultant executable must be incorporated into the parent program without re-compiling it.

Wait on there. It’s a fair ground rule to say that the end-user doesn’t have a Clarion compiler and in any case is not going to get access to precious source. But the application’s developer has (at least one) Clarion compiler. Perhaps expanding a bit on each step in the solution will do the trick:

1. A script is converted into Clarion source. This capability has to be built into the user’s on-site application. Clarion expressions are not the problem, they are already in Clarion source. A code superstructure to hold each expression in its correct logical place must exist, however. In other words, the developer must write a simple pre-processor that converts a complete script into Clarion source and embed this in the end-user’s application.
2. To be legally compiled and linked the source has to find its way back to the developer’s compiler. To automate this process, the developer needs a web site that can receive Clarion source, pass it through to a Clarion compiler for a compile/link and return (preferably) a small DLL to the end-user. A web server that can in principle do this is the Handy Tools Browser Server template, which drops a working web server into a Clarion application. Yes, the developer has to write it in Clarion, but as a result can learn from the source of free batch compilers already written in Clarion. Use of the compiler must be single-threaded, but, with compile and link times expected to be only

a few seconds, this would not be a draw back until the developer was rich enough to solve it.

3. For a remotely compiled function to be executed by the end-user's program (in place of a script), the function's name must already be known to the program. A clean way to do this is for a much shorter script (calling only the new function) to replace the old, much longer one. Any number of already bound dummy functions may be included within the program to be ready and waiting. A shell DLL can also be there to be over-written by the new one. Necessarily the new function will have no parameters. Only variables within the model's data structure can be referred to in a script which is converted to an executable.

Further refinements would allow the end-user to convert more than one script to an executable function. Perhaps an automatic zip/unzip could further speed things up.

Ideally the end-user's app would create Clarion source, contact the developer's web site, and complete a swap of script for executable function in one operation. It could not over-write its own DLL file but could hand this quick job onto another process before being re-executed in a new, speedier incarnation. It is hard to see, however, how replacing the user's old script with the new one could be automated. He or she might just have to do that by hand.

Earlier I said: "To be legally compiled and linked the source has to find its way back to the developer's compiler." This is not strictly true. Because the end-user's app is doing the work of pre-processing a script into standard Clarion source, any owner of a Clarion compiler and an in-house web site could offer the described service.

Well, that's it. Have your script and eat it too. What do you say?

David Podger has been an independent Clarion developer in Australia for a decade. He presently lives in Katherine in the Northern Territory, where he sells a specialised accounting application for remote communities.

Reader Comments

Add a comment

An interesting concept - here are a couple of...
Us Clarion folks tend to always like to do things in...
Thanks to Bruce for his input. His comments, I was...

Clarion Magazine

Translation, Clarion Style

by Nardus Swanevelder

Published 2003-08-07

If you want to make your app available in other languages what do you use and where do you start? If your application is used by various clients in the same industry but your clients have different descriptions for the same concept, what do you do?

One other thing before continuing: in South Africa, where I live, there are 11 official languages of which nine are African languages, one is English and one is a language called Afrikaans. The top four languages in South Africa (by number of speakers) are Zulu, Xhosa, English and Afrikaans. Afrikaans is spoken mainly by the white and coloured people although there are a lot of black people that can speak it as well. Afrikaans is related to Dutch and Flemish but also contains a lot of English, French and German words. English is the business language, so most of the local developed applications are developed in English.

You can create multiple app files and compile multiple EXEs or you can buy one or more products from ProDomus, Pea Brain or the Clarion Handy tools. If you want to do it yourself, you search for more information on translation and you discover that Clarion comes with something called the Clarion Translator class.

Before looking at what the Translator class is, I think it is important to point out that the idea behind this article is not to create an all-inclusive translation solution, but to present a good starting point from which you can grow your knowledge on translation.

What is the translator class? This is what the Clarion online help says:

> By default, the ABC Templates, the ABC Library, and the Clarion visual source code formatters generate American English user interfaces. However, Clarion makes it very easy to efficiently produce non-English user interfaces for your application programs.The TranslatorClass provides very fast runtime translation of user interface text. The TranslatorClass lets you deploy a single application that serves all your customers, regardless of their language preference. That is, you can use the TranslatorClass to display several different user interface languages based on end user input or some other runtime criteria, such as INI

file or control file contents. (Runtime translation is not possible using the standard Clarion ABC template prompts)

Alternatively, you can use the Clarion translation files (*.TRN) to implement a single non-English user interface at compile time.

You can find the Translator Class source files in the Clarion\LIBSRC folder. They are:

- ABUTIL.INC TranslatorClass declarations (also contains INI Manager Class)
- ABUTIL.CLW TranslatorClass method definitions (also contains INI Manager Class)
- ABUTIL.TRN TranslatorClass default translation pairs

Again, from the Clarion help:

> The ABUTIL.TRN file contains translation pairs for all the user interface text generated by the ABC templates and the ABC Library. A translation pair is simply two text strings: one text string for which to search and another text string to replace the searched-for text. At runtime, the TranslatorClass applies the translation pairs to each user interface element.
>
> You can directly edit the ABUTIL.TRN file to add additional translation items. We recommend this method for translated text common to several applications. The translation pairs you add to the Translator GROUP declared in ABUTIL.TRN are automatically shared by any application relying on the ABC Library and the ABC Templates.

An example

The following example shows the necessary steps to use standard Clarion template prompts to translate an application into a foreign language.

Step 1 – Extract the text from your app

The Translator class makes this a breeze. Just follow these easy steps:

1. Open your app and go to the global settings. (I have used the Clarion AutoLog example to test this functionality)
2. Turn **Runtime Translation** on
3. Scroll through the tabs until the **Classes** tab appears.
4. Click on the **General** button and click on the **Configure** button below **TranslatorClass**
5. Enter an **Extract** filename, e.g. AutoLog.trn
6. Close the global settings
7. Before you compile and run the app make sure that all your controls have a use variable; without one Clarion won’t extract the text for those controls, and even if you manually add the text for those controls to the TRN file Clarion will not translate those controls. Check your menu equates as well as your tab equates.

8. Compile and run the application
9. Open all the screens in the application and exit the program
10. On closing the application Clarion will create a sorted file with all the text it found in the application’s screens.

If the application consists of multiple DLLs, the above steps stay the same except that for each DLL you need to supply a different **Extract TRN** filename.

Step 2 – Modify the extracted text

After the TRN file has been created, it has to be checked and corrected before you can continue with the process. But what does a TRN file look like? Here’s a short example:

```
MyAppGroupName  Group
Items USHORT(2)
        PSTRING('Quote')
        PSTRING('')
        PSTRING('Pay')
        PSTRING('')
End
```

The first line indicates that this is a group and I called this group `MyAppGroupName`. The group name is the first line in the file as can be seen in the example above. The next line indicates the number of translation pairs in this group and that is why the value is set to two. The first and third `PSTRING`s are the text exported from the app and the second and fourth lines, currently empty, are where you specify what the replacement text should be.

Now that you know what a TRN file looks like, what should you check for in the TRN file after the export is completed? Look for the following:

1. Single Quotes. If you used text like: "Don’t click on this button" in your application it will be exported like this: PSTRING(‘Don’t click on this button’). If you leave the TRN file like this and you compile your application Clarion will give an error. Just double the quote characters inside the PSTRING to the following: PSTRING(‘Don’’t click on this button’).

   Tip: It is easier to compile the application, and fix the single quote errors using F4 rather than going through the TRN file line item by line item looking for single quote errors.

2. Truncated lines. If your application has a Tip on a Control and there is a carriage return line feed in the Tip, the Tip will be exported in separate lines which will lead to a compiler error. Check the TRN file for truncated lines before compiling. A truncated line will look like this:

   ```
   PSTRING(‘This is a tooltip on
   ```

```
a control’)
```

3. Duplication. In a multi DLL environment some of the same text gets exported to each of your TRN export files. The main menu structure, for example, gets duplicated in all the TRN files.
4. Size of the TRN file. I had a problem when the TRN File had something like twelve thousand lines. Break the TRN file into smaller files and add the different files to the application to bypass this problem. See Step 3, Point 6 (below) for an explanation how to add multiple TRN files to an application.

After the TRN file has been checked it is time to translate the exported text into the language you want.

> **Tip:** Using Excel simplifies editing the TRN file as you can copy and paste lines easily, undo actually works, and it is also easy to calculate the number of string pairs.

Screen design principles for multiple-languages are beyond the scope of this article, but there are resources available on the web that can assist with this. One basic rule is to remember that the translated text should be more or less the same length as the original text otherwise the application screens will have to be redesigned to cater for the variable length prompt text. Even better, design your screens to accommodate languages that tend to require longer words/phrases.

If you want to complete the above steps in source code (still using ABC) you have to add the following lines of code to your application:

Declare an instance of the `TranslatorClass` in Global section of the application:

```
Translator TranslatorClass
```

In the code section at the start of the application, initialize the class and specify the extract TRN file name:

```
Translator.Init
Translator.ExtractText='AutoLog.trn'
```

Just before closing the application you must shut down the class:

```
Translator.Kill
```

And in each procedure after the `Open(Window)` code you have to activate the class by using the following statement:

```
Translator.TranslateWindow
```

The ABC code generator adds the following code to your application before the `Window` is opened:

```
SELF.AddItem(Translator)
```

and it adds the following before a browse:

```
BRW1.Popup.SetTranslator(Translator)
```

To translate a report you will have to do a `SetTarget(Report)` first and then call `Translator.TranslateWindow`.

The ABC code generator adds the following code to your report:

```
SELF.AddItem(Translator)
Previewer.AddItem(Translator)
Translator.TranslateWindow(SELF.Report)
```

Step 3 – Apply the new translated file(s) to your app

The `Translator` class makes it a breeze to apply the TRN files to your application. Just follow these easy steps:

1. Open the app and go to the global settings
2. Turn **Runtime Translation** on (if it is not on already)
3. Scroll to the right until the **Classes** tab appears.
4. Click on **General** and click on the **Configure** button below TranslatorClass
5. Delete the current **Extract** filename (AutoLog.trn). If you leave the **Export** filename in, the application will keep on exporting to this file every time you run the application. This can lead to performance issues depending on the configuration of the application and environment.
6. Click on **Additional Translation Groups** and click on **Insert**.
7. Complete the **Source File Name** and the **Group Label**. This group label must be the same as the group name in the TRN File. (If you didn't delete the extract filename as in point 5, remember to specify a different source TRN filename otherwise your source TRN file will be overridden as soon as you exit your application.
8. Close the global settings
9. Compile and run the application
10. The application will now use the new language.

Multiple DLLs

If the application consists of multiple DLLs, the above steps stay the same except each DLL needs to have a different TRN Source filename and a different Group Label.

If you are using the ABC chain and for what ever reason do not want to implement the

Translator class by using the template prompts the following lines of code need to be added to your application:

In the Global section of the application the class needs to be instantiated:

```
Translator TranslatorClass
```

In the code section at the start of the application the class needs to be initialized. Also add the translation group.

```
Translator.Init
Translator.AddTranslation(AutoLog)
```

The `AutoLog` parameter that is used in the above method is the group name that was specified in the TRN file. If you are using the example TRN file that I described earlier, then add the following code:

```
Translator.Init
Translator.AddTranslation(MyAppGroupName)
```

Just before closing the application the class needs to be shut down:

```
Translator.Kill
```

And in each procedure after the `Open(Window)` code you have to activate the class by using the following statement:

```
Translator.TranslateWindow
```

That will initiate translation.

There's still work to be done, however. Next week I'll look at how to translate standard Clarion messages.

Nardus Swanevelder was born and raised in South Africa. He was a networking engineer for seven years before he moved over to the commercial side of the business. Nardus has developed a Sale Cycle Management system for the Information and Communication Technology industry. Nardus has been programming in Clarion since 1989, and holds B.Com and MBA degrees.

Reader Comments

Add a comment

Clarion Magazine

A C6 Tagging Class Template Wrapper

by Jim Katz

Published 2003-08-07

I left off in the first installment having developed the template to the point that it would generate the necessary code to replace most of Steve's hand coded embeds to set the Tag class in motion. The morning after I wrote that code, while having breakfast with my local biker friends down at Obi's, one guy asked me, " Do you ever have those times when you put the code to bed everything is OK, but in the morning the darn thing is broken?" I said not recently, since the first part of this template exercise was working OK. However when I got home and sat down at the computer I realized I was still missing a couple of pieces to the ABC puzzle.

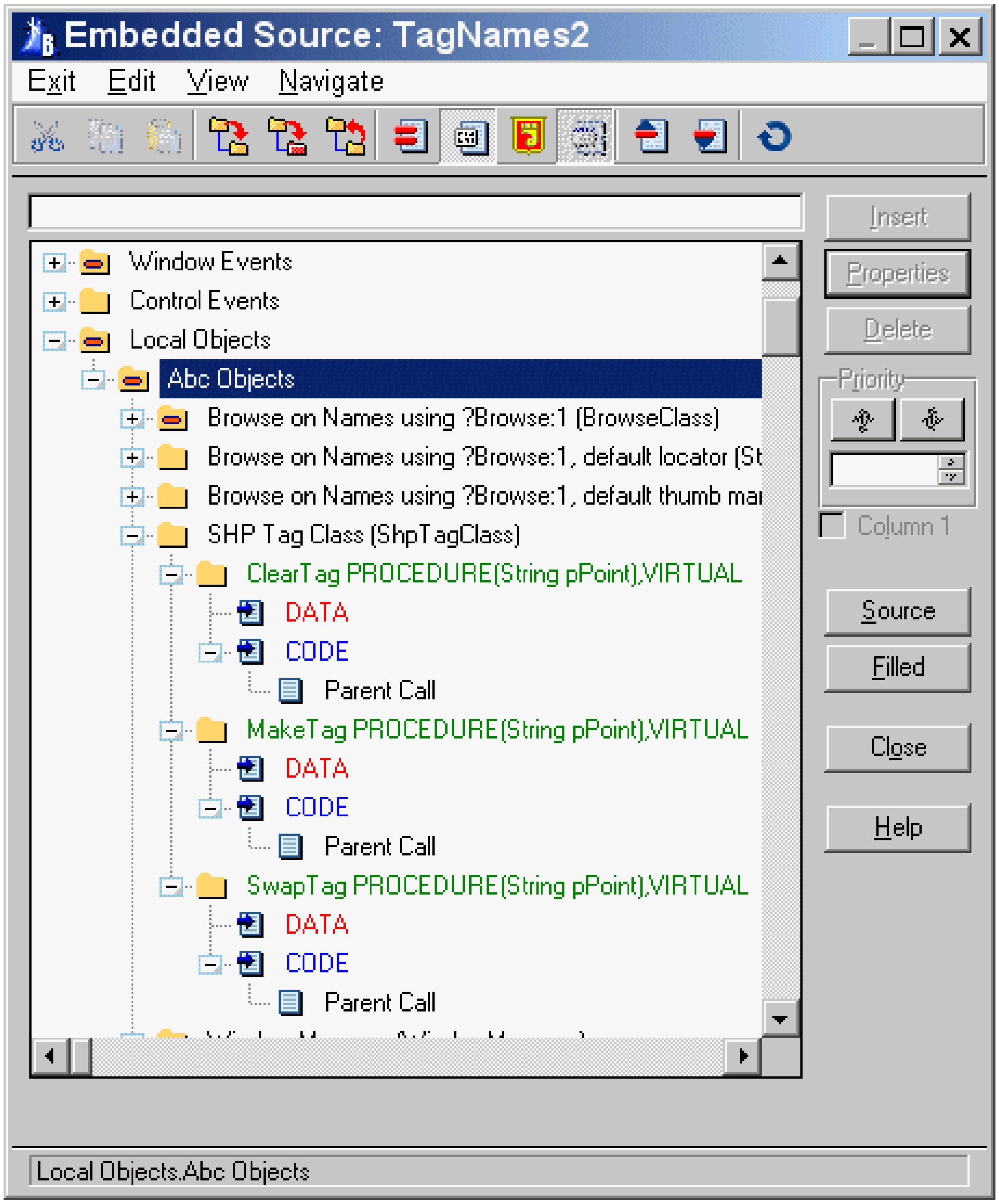
When I looked at the `SHPTagClass` in the ABC Viewer, I noticed two things: there were no `VIRTUAL` methods, and the Constructor and Destructor methods were in essence `PUBLIC`. So I set the `PRIVATE` attribute on those two methods and added the `VIRTUAL` attribute to `MakeTag`, `ClearTag` and `SwapTag`, to make them accessible to the developer. Now I needed to get those `VIRTUAL` methods to show up in the embed tree. To accomplish this I used the ABC group `%GenerateVirtuals` like this:

```
#AT(%LocalProcedures)
#CALL(%GenerateVirtuals(ABC), 'ShpTagClass', ↵
    'Local Objects|Abc Objects|SHP Tag Class',
'%JKSHPTagClassVirtuals(JKSHPTagClassTemplate)')
#ENDAT
```

This places my `VIRTUAL` methods in the embed tree under **Local Objects; ABC Objects** as the **SHP Tag Class**, and uses the group `%JKSHPTagClassVirtuals` to define the **Data** and **Code** method sections and appearance.

```
#GROUP(%JKSHPTagClassVirtuals, %TreeText, ↵
    %DataText, %CodeText)
#EMBED(%JKSHPTagClassDataSection,↵
    'JK SHP Tag Class Method Data Section'),↵
    %ActiveTemplateInstance,%pClassMethod,↵
    %pClassMethodPrototype,LABEL,DATA,↵
```

```
      PREPARE(CALL(%FixClassName,%FixBaseClassToUse('SHPTagClass'))),↵
      TREE(%TreeText & %DataText)
    #?CODE
    #EMBED(%JKSHPTagClassCodeSection,↵
      'JK SHP Tag Class Method Executable Code Section'),↵
      %ActiveTemplateInstance,%pClassMethod,%pClassMethodPrototype,↵
      PREPARE(CALL(%FixClassName,%FixBaseClassToUse('SHPTagClass'))),↵
      TREE(%TreeText & %CodeText)
  #!
```



```
+ Window Events
+ Control Events
- Local Objects
  - Abc Objects
    + Browse on Names using ?Browse:1 (BrowseClass)
    + Browse on Names using ?Browse:1, default locator (St
    + Browse on Names using ?Browse:1, default thumb mai
    - SHP Tag Class (ShpTagClass)
      - ClearTag PROCEDURE(String pPoint),VIRTUAL
          DATA
        - CODE
            Parent Call
      - MakeTag PROCEDURE(String pPoint),VIRTUAL
          DATA
        - CODE
            Parent Call
      - SwapTag PROCEDURE(String pPoint),VIRTUAL
          DATA
        - CODE
            Parent Call
```

Local Objects.Abc Objects

Figure 1. The generated embeds

Now I also needed to generate a parent call for each of the `VIRTUAL` methods. Here's the code:

```
#AT(%JKSHPTagClassCodeSection),PRIORITY(5000),↵
    DESCRIPTION('Parent Call')
  #CALL(%GenerateParentCall(ABC))
#ENDAT
```

Finally, I needed to create the local declarations for the derived class methods in the procedure data area, which the ABC group `%GenerateClass` makes easy:

```
#AT(%LocalDataClasses)
  #IF(%JKShpTagClassScope='Local')
#INSERT(%GenerateClass(ABC), 'ShpTagClass')
  #ENDIF
#ENDAT
```

Note that while I can cause the same code to generate `ShpTagClass` globally, it does present other problems that are beyond the scope of this article. That is, if I declare the derived methods globally I have to cause the method code to be generated in the same scope, in the main program module. As a result, the globally declared tag class will have to use the class code as written. with no derived or new methods.

Summary

I began this template and article as a personal challenge after reading a news group exchange on the subject of a Clarion 6 compliant tagging solution. Steve Parker offered his class, and I thought that I could readily put together the necessary template bits to make it an ABC template as well. I had gone thru this exercise, initially, five years ago when the first ABC template chain was released for Clarion 5. It's nice to see how much the template code has been cleaned up in C6, with most of the critical code now in ABC `#Groups`.

Download the source

Jim Katz has been working in Clarion since 1987, when he discovered version 1.1 for DOS. At that time he was developing internal applications for his family business, the BMW motorcycle franchise in Daytona Beach, FL. It was quite a thrill when CPD 2.0 came out with the Designer. Jim modified the table model to allow reverse sorting, and this modification was offered on one of the freeware tool diskettes at the time as REVRSTBL.MOD. In 1993 Jim left the motorcycle business to work full time as a programmer for a small computer shop in DeLand, Florida, using CPD 2.1 and LPM, and then in 1996 became an independent contractor. In 1998 he took over the maintenance of the DET/DEF products originally developed by Tom Moseley. Jim's current project is an internet enabled multi-player board game with the client in Java and the game server engine in Clarion. He currently resides on ten secluded acres in rural Deleon Springs, FL with two cats and an impressive number of wild critters.

Reader Comments

Add a comment