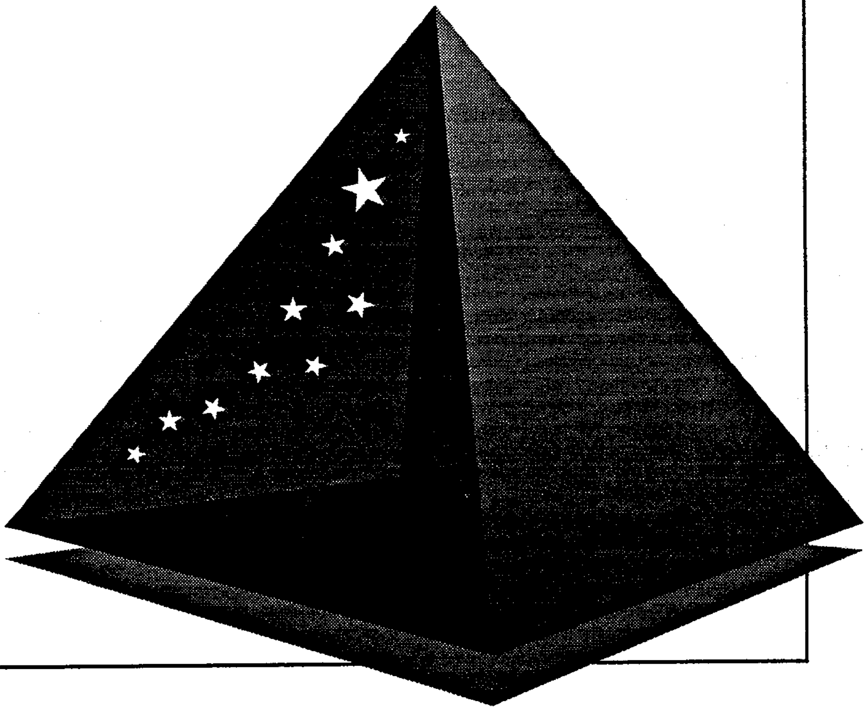
DEVCON '94

INTEGRATING TOPSPEED WITH CLARION

Marcia Holmes



TopSpeed™ CORPORATION

Integrating Clarion & C

Marcia Holmes
DevCon '94

Abstract:
======

. Why use C with Clarion

. How to get Clarion Database Developer and C to work together

. How to get Clarion for Windows and C to work together

. Examples of Integrating Clarion with C

****This discussion will spend most of the period studying practical examples.

Assumptions:
==========

. Knowledge of CDD

. Some understanding of C

Introduction

=========

[illegible]

Clarion Database Developer v3.0 is an *exceptional* application development tool. It is possible to generate in a few short hours, an application that would take weeks or months to code from scratch in C. This begs the question of why we might want to use C at all within a Clarion application, particularly as Clarion 3.0 has a code generation capability that rivals that of many C compilers. The answer isn't necessarily straightforward, the majority of Clarion developers will continue to develop larger and increasingly complex applications without ever having the need link to external C routines.

There is a wealth of programming libraries available which may ease the task of developing complex applications. Statistical, financial, graphics and communications libraries are readily available, any one of which could significantly cut the development time of a typical application. The majority of these programming tools are developed in C, and it is also for this reason that the capability of interfacing Clarion and C is so important.

C provides low-level capabilities that prove useful to Clarion programmers:

*Allows access to interrupts that are not available via int86
*Has the cleanest interface to the DOS Extender API functions
*Handles functions that return pointers with ease

Running Clarion 3 and TopSpeed together

Clarion Database Developer ver 3 (CDD3) and DOS Protected Mode TopSpeed version 3.1 (TS3.1) are designed to work together.

The real mode version of TopSpeed does not work with CDD3. If you have this system, you cannot also have Clarion3 on your path when doing TopSpeed work.

Although CDD and TS3.1 are designed to work together, there is some work to be done to get them to work together nicely. This article assumes that Clarion is installed in `C:\CLARION3` and TopSpeed in `C:\TS`.

The first thing necessary to get CDD and TS3.1 working is to have your path set to point at both products.

CLARION3 MUST BE BEFORE TOPSPEED ON THE PATH!!

So your path would look something like:

```
    C:\CLARION3;C:\TS\XTD_SYS;C:\TS\SYS;C:\TS\OS2_SYS
```

See the addendum on 'How to shrink your path when using TopSpeed' .

CDD and TS3.1 use a redirection system to find the location of files. With your path set as above, you will be using the redirection file supplied with CDD, which is not aware of your TopSpeed products. To make it aware you need to copy the contents of `C:\TS\XTD_SYS\TS.RED` to the end of `C:\CLARION3\CLARION.RED.`

You can do this in any way you like. However, here is how to do it via the TopSpeed environment:

```
  .  CD C:\TS\XTD_SYS          -- change to the TopSpeed directory
  .  TS                        -- start TopSpeed
  .  Alt-S,E                   -- Load redirection file for edit
  .  Alt-1 TS.RED              -- Load TS.RED into window 1
  .  Ctrl-K B                  -- Highlight TS.RED
  .  Alt-9                     -- Goto CLARION.RED
  .  Ctrl-PgDn                 -- Goto the bottom of CLARION.RED
  .  Ctrl-K G <Enter>          -- Copy the contents of Window 1
  .  F2                        -- Save the new CLARION.RED
  .  Alt-X                     -- Exit TopSpeed
```

Now you can compile from either the TopSpeed or Clarion IDEs. From TopSpeed, entering `#compile mycode.cla` in your PR file will compile your Clarion code. Adding mycode.c as an external source file in your Clarion project would cause your C code to compile from the Clarion IDE.

One extra note for people creating TopSpeed libraries for use with Clarion: To get the project system to use the clarion libraries, you need to add the line `` `#set tscla=on' `` to your project file somewhere before the `#link` line. This is not necessary if you are compiling any clarion code in the project.

Linking TS 3.1 into CW and TS 3.1

======================

This requires Clarion for Windows (of course) the TopSpeed protected mode C compiler (v3.1) and the Techkit (which now ships with the C/C++ pack).

The project file for your TopSpeed program must specify the windows system (win) and memory model *large*.

In your .CLW program, you must prototype the TopSpeed functions you will use appropriately following the guidelines in the Clarion API Toolkit.

One restriction must be noted: You CANNOT link TopSpeed C as external source. You must link it as an .OBJ or .LIB in the .PRJ file of your .CLW program.

If you want to be able to debug CW and TS code together, you cannot build the TS code as a library and link the library (debug information is not stored in a library). You can link in the separate .OBJ files or build a .DLL and link in the matching .LIB file (debug information is stored in a DLL). You must have the C source code and the .DBD for the C object in a directory that the debugger can find (based on the redirection file)

The basic idea is that you look for the last character of the search string A in the buffer B. You know that it has to be at Start+SIZE(A)-1 for it to be part of A in B. If it is you look backwards to see if you have a match. If the character found is not the last character of A, then you can skip along the number of characters that would put the found character in A. For example. If looking for 'Fred' in 'This is a test to see if Fred can be found' starting at position 1 you would first get the fourth character ('s'). 's' is not a character in 'Fred', so you know you can skip another 4 characters, giving ' '. Again ' ' is not in 'Fred', so you skip 4 characters giving 'e'. Now this is not 'd', but it is in 'Fred' so we can only go 1 character, as 'e' is the character before 'd' in 'Fred'. This gives 's', which is not in 'Fred', so we can skip 4 characters again, giving 'o'. Skipping 4 then gives 'e' again. Skipping 1 gives ' '. Skipping 4 gives 'F'. Skipping 3 gives 'd', which is what we are looking for. Going backwards finds 'Fred' and we stop, returning 26, the offset of 'Fred' in the string.

The search function return the offset of string A in B. It returns 0 if no match is found. The search algorithm is based on the Boyer-Moore algorithm. It is limited to search strings of 512 chars or smaller.

Test.CLA

```
PROGRAM
         MAP
           TimeSearch(STRING,LONG)
           DisplayTime (STRING,STRING,LONG,LONG)
           MODULE('Search')
             AIsInB(STRING,STRING,LONG),LONG
                 .
               MODULE('CSearch')
             CAIsInB(STRING,STRING,LONG),LONG
           .
         .
LongString      STRING(10000)
Parts           GROUP,OVER(LongString)
Start             STRING(7)
                  STRING(4000)
Middle            STRING(10)
                  STRING(5000)
EndBit            STRING(7)
                .
Count           USHORT,AUTO
        CODE
          ! Fill the search string with characters 0 to 9
          LOOP Count = 1 TO 10000
            LongString[Count] = Count%10
          .
          ! Add Some search Text to the beginning of the search string
          Start = 'A Fred '
          ! Add Some search Text to the middle of the search string
          Middle = 'MiddleText'
          ! Add Some search Text to the end of the search string
          EndBit = 'EndText'
          TimeSearch('Fred',1)
          TimeSearch('A',1)
          TimeSearch('This long string does not exist',1)
          TimeSearch('No',1)
          TimeSearch('8901234566',1000)
          TimeSearch('EndText',1000)
          TimeSearch('MiddleText',1)
```

```
TimeSearch      PROCEDURE (SearchString,OffSet)
SubPosition     LONG,AUTO
StartTime       LONG,AUTO
StopTime        LONG,AUTO

        CODE
          StartTime = CLOCK()
          LOOP 100 TIMES
            SubPosition = INSTRING(SearchString,LongString,1,OffSet)
          .
          StopTime = CLOCK()
          DisplayTime ('INSTRING',SearchString,SubPosition,StopTime-StartTime)
          StartTime = CLOCK()
          LOOP 100 TIMES
            SubPosition = AIsInB(SearchString,LongString,OffSet)
          .
          StopTime = CLOCK()
          DisplayTime ('AIsInB  ',SearchString,SubPosition,StopTime-StartTime)
          StartTime = CLOCK()
          LOOP 100 TIMES
            SubPosition = CAIsInB(SearchString,LongString,OffSet)
          .
          StopTime = CLOCK()
          DisplayTime ('CAIsInB ',SearchString,SubPosition,StopTime-StartTime)
          RETURN

DisplayTime     PROCEDURE (Func,SearchString,Pos,TimeTaken)
          CODE
            TYPE (Func&' ')
            IF Pos > 0
              TYPE ('found '&SearchString&' at position '&Pos)
            ELSE
              TYPE ('did not find '&SearchString)
            .
            TYPE (' in '&TimeTaken&' seconds'&@LF)
            RETURN
```

Search.CLA

```
PROGRAM

          MAP

            MODULE('Core')

             memset(*STRING,SHORT,USHORT),RAW,NAME('_memset')


                    memcmp(*STRING,*STRING,USHORT),SHORT,RAW,NAME('_memcmp')

            .

            AIsInB(STRING,STRING,LONG),LONG

          .

          INCLUDE ('CLARION.EQU')



            CODE


AIsInB      FUNCTION(A,B,Start)


Offsets     STRING(256),AUTO

AOffset     BYTE,AUTO

BOffset     BYTE,AUTO

LastChar    BYTE,AUTO

LoopCount   USHORT,AUTO

Count       USHORT,AUTO


            CODE

              ! Initialise Offsets to hold a default jump value of

              ! the size of A.  Set all offsets for elements of A

              ! to the smallest jump necessary for the character to be

              ! in A.  For example, the jump for 't' in 'Brigitta' is 1.


              memset (Offsets,SIZE(A),SIZE(Offsets))


              LOOP LoopCount = 1 TO SIZE(A)-1

                AOffset = VAL(A[LoopCount])

                Offsets[AOffset+1] = CHR(SIZE(A)-LoopCount)

              .
```

```
Count = SIZE(A)+Start-1
LastChar=VAL(A[SIZE(A)])
LOOP UNTIL Count > SIZE(B)
  IF VAL(B[Count]) = LastChar
    ! Use the fast C routine to compare A with the preceeding
    ! characters for a match.
    IF 0 = memcmp(A,B[Count-SIZE(A)+1],SIZE(A))
      RETURN (Count-SIZE(A)+1)
    .
  .
  BOffSet = VAL(B[Count])
  Count += VAL(Offsets[BOffSet+1])
.
RETURN(0)   ! If we got this far then no match was found
```

CSearch.C

```
#pragma warn(wall=>on)
#include <memory.h>
#pragma name(prefix=>"")
long CAISINB(unsigned alen, char *a, unsigned blen, char*b, long start)
{
  char offsets[256];
  char lastchar;
  unsigned j;
  long     i;

  memset (offsets,alen,sizeof(offsets));
  for (j=0;j<alen-1;j++)
    offsets[a[j]]=alen-j-1;
  i=start+alen-2;
  lastchar=a[alen-1];
  while (i<=blen)
  {
    if (b[i]==lastchar)
    {
      if (!memcmp(a,&b[i-alen+1],alen))
        return (i-alen+2);
    }
    i+=offsets[b[i]];
  }
  return (0);
}
```

Protected mode version of OpenSem function
Novell(c) functions for Clarion(c) interface to network calls

```
*pSem_Name - pointer to semaphore name
 nInit_Value - semphore number

 Returns Number of workstations with the passed semaphore
         or
 Returns FALSE (negative number) if unsucessful.

. Name passed a STRING.
```

Summary of function call:

On entry:

```
AH = 0xC5
AL = 0x00
DS:DX = Pointer to the semaphore name string
        byte zero has length of string 1-127
CL = Initial semaphore value
```

On return:

```
AL = Completion code
      0x00 = Created OK
      0xFF = Invalid initial value
      0xFE = Invalid string length
      0x96 = Out of work memory in file server
   BL = Number of stations that have this semaphore open
   CX,DX = Semaphore handle
```

ClaSem.CLA

```
PROGRAM
          MAP
            MODULE('Csem')
             pOpenSem(STRING,SHORT),SHORT,NAME('_pOpenSem')
             .
          .

pName   STRING(10)
nValue  SHORT
nRvalue SHORT

        CODE
           pName   = 'MYSEM'
           nValue  = 1
           nRvalue = pOpenSem(pName,nValue)
           TYPE('Return value is '&nRvalue)
           ASK
```

PopenSem.C

```
#include <tsxlib.h>
#include <dos.h>
#include <string.h>
#include <stdlib.h>


#define OPEN_SEMAPHORE  0xC500          /* Defines */
#define BLOCKSIZE       128

#ifdef _XTD
int pOpenSem(unsigned Sem_len,char *pSem_Name, int nInt_Value)
{
    unsigned int SelValue ;
    long Address ;

    struct REGPACK r ;

    int Size = 0 ;
    char cBuffer[BLOCKSIZE] = "" ;     /* Buffer for string */
    char *pAddress ;                   /* Pointer for address */
    void far *pLow ;

//    Size = strlen(pSem_Name);         /* Get size */
    Size = Sem_len;
    itoa( Size, cBuffer, 10);          /* Convert to Char type */
    pAddress = strcat(cBuffer, pSem_Name);        /* Add semaphore name */

    SelValue = ALLOCLOWSEG(BLOCKSIZE) ;  /* Allocate Low Memory for Real Call */
    pLow = MK_FP(SelValue, 0) ;          /* Make Far Ptr With Selector Value */
    strcpy(pLow,cBuffer );             /* Move Data Into Low Memory */

    Address = MAKEREALADDR( SelValue, 0) ;  /* Make a Real Adr for Real Call */

    r.r_ax = OPEN_SEMAPHORE ;       /* Set function 0xC5 */
    r.r_cx = nInt_Value ;           /* Set semaphore number */
    r.r_dx = FP_OFF(Address) ;      /* Set Real Address Offset Value */
    r.r_ds = FP_SEG(Address) ;      /* Set Real Address Segment Value */


    REALINTR(&r,0x21) ;
    FREESEG(SelValue) ;

    return r.r_bx ;

}
```

```
#else
#if 0
CLASHORT pOpenSem(char *pSem_Name, CLASHORT nInt_Value)
{
    struct REGPACK r = {0};

    int Size = 0 ;
    char cBuffer[BLOCKSIZE] = "" ;     /* Buffer for string */
    char *pAddress ;                   /* Pointer for address */

    Size = strlen(pSem_Name) ;         /* Get size */
    itoa( Size, cBuffer, 10);          /* Convert to Char type */
    pAddress = strcat(cBuffer, pSem_Name);        /* Add semaphore name */

    r.r_ax = OPEN_SEMAPHORE ;      /* Set function 0xC5 */
    r.r_cx = nInt_Value ;          /* Set semaphore number */
    r.r_dx = FP_OFF(pAddress) ;     /* Set Real Address Offset Value */
    r.r_ds = FP_SEG(pAddress) ;     /* Set Real Address Segment Value */

    intr(0x21, &r) ;

    return r.r_bx & 0xFF;
}
#else
#define TRUE 1
#define FALSE 0

CLASHORT pOpenSem( char *pSem_Name, CLASHORT nInit_Value)
    {

     union REGS regs = {0};                   /* Locals  */
     union REGS inregs = {0};
     struct SREGS sregs= {0};
     struct SREGS segs= {0};
     int Size = 0 ;
     char cBuffer[128] = "" ;           /* Buffer for string */
     char *pAddress ;                   /* Pointer for address */


     cBuffer[0] = (unsigned char)strlen(pSem_Name) ;         /* Get size */
     pAddress = strcat(cBuffer, pSem_Name);        /* Add semaphore name */
                                        /* strcat returns pointer to address */

     inregs.x.ax = OPEN_SEMAPHORE ;     /* Set function 0xC5 */
     inregs.h.cl = nInit_Value ;        /* Set semaphore number */
     inregs.x.dx = (unsigned int)pAddress ;    /* Set string address */

     segread(&sregs) ;                  /* Read segment registers */

     sregs.ds = sregs.ss ;              /* Set DS to SS */

     intdosx( &inregs, &regs, &sregs ); /* Call int21h */

     if(regs.h.al)                      /* Test AL */
     {    printf("al = 0x%02X\n", regs.h.al);
          return -regs.h.al ;           /* Return error code; negated */
     }

     if(!regs.h.bl)                     /* Test BL for number of workstations */
     {       printf("bl = 0\n");
             return FALSE ;
     }

     printf("Call OK\n");
     return regs.h.bl ;                 /* Return number of semaphores open */

     }

#endif
#endif
```

Addendum

How to shrink your path when using TopSpeed

When using the TopSpeed DOS Extended Environment and Protected mode compiler you have to set your path to `C:\TS\XTD_SYS;C:\TS\SYS;C:\TS\OS2_SYS`. This uses up a lot of valuable path space. The following text to explains why is is this way and what you can do to reduce your path.

TopSpeed comes in three forms:

TopSpeed for DOS
TopSpeed for Protected mode DOS
TopSpeed for OS/2.

There used to be only TopSpeed for DOS and TopSpeed for OS/2. Then you had `TS\SYS` for DOS and `TS\OS2_SYS` for OS/2. When TopSpeed introduced the DOS Extended system, it was a murger of these two systems. It used some of the EXEs from the DOS product, the DLLs from the OS/2 product and added some EXEs of its own.

Hence the need for the three paths:

`XTD_SYS` holds the new EXEs
`OS2_SYS` holds the protected mode DLLs
`SYS` holds the DOS EXEs

So, if you are not using TopSpeed for OS/2 or the real mode TopSpeed system you can reduce your path to one entry `TS\SYS` by:

. Copying all .DLL files from `OS2_SYS` to SYS
. Copying all of `XTD_SYS` to SYS

You can then change your path to just `TS\SYS`.