

---

## Chapter 4 Compilers & Development Environments

---

Ramblings about MacApp.....	68
To all users of ThinkC and MultiFinder.....	69
MPW Pascal and self referencing.....	70
CClusters in Think C.....	71
LSP 3.0 bug (?).....	71
global data in Think C.....	72
THINK Pascal interfaces (was SysEnvirons).....	72
Alternative pkg interfaces for Pascal.....	73
Bug in THINK Pascal? My bug?.....	77
A possible bug in THINK C 4.0.1.....	77
32k arrays (in Think Pascal).....	78
Multiple Inheritance for HandleObjects in C++.....	79

---

## USENET Macintosh Programmer's Guide

---

From: keith@Apple.COM (Keith Rollin)  
**Subject: Re: Ramblings about MacApp**

In article <1989Nov2.190025.14568@polyslo.CalPoly.EDU> tdrinkar@cosmos.acs.calpoly.edu.UUCP (Terrell Drinkard) writes:

```
>New subject:
> Can someone out there give me some idea as to the usefulness of
>MacApp? I'm not even real sure as to what it is, actually. But it
>was noted in my MPW docs and I am interested.
```

Let me put it this way: I won't program without MacApp anymore. And I know that anyone else in DTS who had mastered MacApp and needs to write a real program feels the same way (unless your name is Paul, in which case you'll use ACL).

MacApp is an object oriented application framework. It is written in Apple's Object Pascal, but can also be accessed by C++. It gives you the basic frame- work of an application, and allows you to fill in the missing parts. For instance, all you need to do to get a working program that supported a full set of menus, multiple windows, window management, about box, scrollbars, memory management, exception handing, multifinder awareness, etc. is write about 50 lines of source code. Here is a little Nothing program that comes with MacApp that gives you all that I mentioned. Basically, all it does is create an Application object responsible for running your program (from the main event loop on down), and defines a procedure to be called when the window needs to be updated (TDefaultView.Draw):

```
program UNothing;

uses
  UMacApp, UPrinting, Fonts;

const
  kSignature = 'SS01';
  kFileType = 'SF01';

type
  TNothingApplication = object(TApplication)
    procedure TNothingApplication.INothingApplication (itsMainFileType: OSType);
    end;

  TDefaultView = object(TView)
    procedure TDefaultView.Draw (area: Rect);
    OVERRIDE;
    end;

var
  gNothingApplication: TNothingApplication;

procedure TNothingApplication.INothingApplication (itsMainFileType: OSType);

begin
  IApplication(itsMainFileType);
  RegisterStdType('TDefaultView', 'dflt');
  if gDeadStripSuppression then
    if Member(TObject(nil), TDefaultView) then
      ;
  end;

  procedure TDefaultView.Draw (area: Rect);
    OVERRIDE;

    var
      itsQDExtent: Rect;

  begin
```

```
PenNormal;  
PenSize(10, 10);
```

---

## USENET Macintosh Programmer's Guide

---

```
PenPat (dkGray);
GetQDExtent (itsQDExtent);
FrameRect (itsQDExtent);
TextFont (ApplFont);
TextSize (72);
MoveTo (45, 90);
DrawString ('MacApp (');
PenNormal;
end;

begin

InitToolBox;
if ValidateConfiguration (gConfiguration) then
begin
    InitUMacApp (8);
    InitUPrinting;
    New (gNothingApplication);
    FailNIL (gNothingApplication);
    gNothingApplication.INothingApplication (kFileType);
    gNothingApplication.Run;
end
else
    StdAlert (phUnsupportedConfiguration);
end.
```

In addition, MacApp gives you excellent development tools and debugging facilities, including an object inspector, a class browser, a view layout editor, discipline, writeln window, high-level breaks and tracing, performance tools, etc.

> And if you don't mind, could someone also explain the benefits  
> of belonging to APDA?

You can buy MacApp...

-----  
Keith Rollin --- Apple Computer, Inc. --- Developer Technical Support

●●●

From: u-atgoat%ug.utah.edu@cs.utah.edu (Alan T Goates)

**Subject: To all users of ThinkC and MultiFinder...**

Keywords: Think C, MultiFinder, ResEdit

Don't you hate it when you have Think C already running, and you try to double click a project from the Finder. No dice, MultiFinder gives you an error message. Well, I got to poking around a bit and fixed this problem. Here's how:

First, turn MultiFinder off and re-boot (if MF is on).  
Then run ResEdit and open up MultiFinder.  
Open up the 'mst#' resources.  
Create a new one, and give it ID #102  
Open it up as a 'STR#' type.  
Add the two strings "File" and "Project".  
Open up 'mst#' ID 101 as a 'STR#'.  
Add the string "Open Project..."  
make sure the the ... is just one character (option-semi-colon).  
Now go into the 'mstr' type and delete ID #102.  
Save and re-boot with MultiFinder on.

That's it. Have Fun.

AL

• • •

---

## USENET Macintosh Programmer's Guide

---

From: keith@Apple.COM (Keith Rollin)

**Subject: Re: MPW Pascal and self referencing**

In article <939@uvicctr.UVic.CA.UUCP> franklin@uvicctr.UUCP (Katherine Franklin) writes:

```
>I am writing an MPW Pascal tool, and I have written an invariant check
>function. What I would like to do is call this function many times
>throughout the program and each time the invariant *doesn't* hold is to
>write a message to the console.
>
>Unix C has a macro variable that allows you to pass the current line number
>in the source code on to the object code to report during execution. I would
>like to do this, or some other dynamic way.
>
>Is this possible ? if so, how ?
>
>      Thanks,
>      Katherine
```

Katherine,

I don't know if there is a way to get the actual line number an instruction came from in MPW Pascal. However, there is a kludge that will give you the procedure name as long as you have compiled your program with Debugger names turned on. MacApp uses the following routine in its own debugger to determine where errors occurred, or for monitoring tracing:

```
PROCEDURE GetProcName(ppc: Longint;VAR className, procName: MAName);
{ GetProcName returns the name of the procedure or function in
  which ppc points.  If it is in a method, then it return's
  the name of the method's class in className. }

VAR
  pc, nextPC, limit:  Ptr;
  index:              INTEGER;

BEGIN
  pc := Handle(ppc)^;
  IF (ord(pc) <> 0) & NOT Odd(ord(pc)) THEN
    BEGIN
      limit := Ptr(ord(pc) + 32767);
      WHILE (endOfModule(pc, limit, @procName, nextPC) = NIL) DO
        BEGIN
          IF ord(pc) >= ord(limit) THEN
            BEGIN
              className := '';
              procName := '';
              LEAVE;
            END
          ELSE
            pc := Ptr(ord(pc) + 2);
        END;

      index := pos('.', procName);
      IF index <> 0 THEN
        BEGIN
          className := copy(procName, 1, index - 1);
        END
      ELSE
        className := '';
      END
    END
  ELSE
    ;
END;
```

```
BEGIN  
className := '';  
procName := '';
```

---

## USENET Macintosh Programmer's Guide

---

```
END;  
END;
```

("endofmodule" is in the DisAsm library that comes with MPW.) From your assertion routine, call this routine with a pointer to the return address of the routine that called your assertion. You can get this return address with something like the following expression:

```
Ord4 (GetCurStackFramePtr) + 4
```

Where "GetCurStackFramPtr" is an inline procedure that looks like:

```
FUNCTION GetCurStackFramePtr: Ptr;  
    INLINE $2E8E;                { MOVE.L A6, (A7) }
```

Hope this helps,

-----  
Keith Rollin --- Apple Computer, Inc. --- Developer Technical Support

•••

From: lim@iris.ucdavis.edu (Lloyd Lim)  
**Subject: Re: CClusters in Think C**

In article <468@adimail.UUCP> tel@adimail.UUCP (Terry Monks) writes:

```
>[last time I asked such a question here I got only 3 responses, of which only  
>1 was of any value - which makes me think that not many people are actually  
>using Think C objects...]  
>  
>If I have a CCluster of objects, all of which have a PrintMe method, how do  
>I get the whole group to PrintMe? If I use DoForEach, it seems to me that I  
>need an intermediate routine that does nothing but call PrintMe for each  
>object passed, and that can't be efficient, can it?
```

Yes, the way it is currently set up you have to have a short little static routine to pass to DoForEach. I commented on how this seemed inelegant in a previous thread and Tom Lippincott replied with a cleaner way. After thinking about it a little bit, both methods are about the same in efficiency but Tom's suggestion doesn't need those static routines everywhere. Go back and look for his reply under the subject Re: OOP & algorithms if you are interested.

Lloyd Lim    Internet: lim@iris.ucdavis.edu (128.120.57.20)

•••

From: hal@krishna.cs.cornell.edu (Hal Perkins)  
**Subject: Re: LSP 3.0 bug (?)**

In article <XCFSY@xavier.swarthmore.edu> Nick Jackiw writes:

```
>Here's an annoying discrepancy I turned up in THINK Pascal (v3.0), which  
>can cause run-time behavior in the environment to differ from behavior  
>in a compiled application.
```

[sample code whose behavior changes]

```
>      with aPtr^ do  
>      aPtr:=Ptr(ord(aPtr)+x+y+z)  
>  
>where x, y, and z are fields of whatever aPtr pointed to, they may or
```



>may not be valid when the compiler sums the r-expr.  
>  
>I'm not sure I'm correct in saying this is a \*bug\*--I don't know what  
>the Pascal definition says in this case--but if it's not a bug in

---

## USENET Macintosh Programmer's Guide

---

>pascal, it might be a bug in your code, so watch out.

The only bug here is that whoever wrote this code doesn't know Pascal well enough. Standard Pascal says that in the statement

```
with e do s
```

the statement `s` may not modify any of the variables referenced in the expression `e`. This is precisely to permit the compiler to optimize the statement by keeping the pointer expression in a register or whatever else is helpful.

Hal Perkins                      hal@cs.cornell.edu

●●●

From: ech@cbnewsk.att.com (ned.horvath)

**Subject: Re: global data in Think C**

From article <walrjuG00UgyAPK458@andrew.cmu.edu>, by cc4b+@andrew.cmu.edu (Christopher Brian Cox):

> Rich,  
>  
> Face it, the 32k global data limit is a BUG!  
> I have a little program here called p2c. It had 90+k of global data  
> when I first compiled it. After removing the un-initialized arrays  
> and allocating them at runtime, it had 48+k of global data.  
> That's with Seperate STRS.  
> The compiler should automatically allocate un-initialized arrays at  
> runtime. There still shouldn't be a 32k limit.  
>  
> Chris Cox

Ease off. 32K is a restriction, an irritation at worst. It is rooted in the fact that the MC68000 uses 16-bit signed offsets for address-register based addressing, and furthered by Apple's decision to use a program model that includes "globals are accessed as negative offsets from a5."

There are ways around this for the compiler writer, but because of the inherent limitations of the processor chip, they all require explicit address arithmetic in the generated machine code. That means larger, slower programs.

You have a couple of alternatives. One you've identified: using NewPtr or NewHandle at run time for uninitialized arrays cut your "globals" requirement in half. That has the rather nice side effect that you can resize such arrays to match the problem size.

For large initialized arrays, consider "compiling" them into resources, or reading them in from your data fork. I won't argue that this is convenient, and you might justifiably claim that Motorola, Apple, and Think/Symantec have made your life a little tougher. The only compiler I know of that lets you duck the limit (for a price in speed and size) is Aztec C.

=Ned Horvath=

●●●

From: gft\_robert@gsbacd.uchicago.edu

**Subject: THINK Pascal interfaces (was SysEnvirons)**

In article <1043@gargoyle.uchicago.edu>, dawyd@gargoyle.uchicago.edu (David Walton) writes...

>In article <41411@apple.Apple.COM> anderson@apple.COM (Clark Anderson) writes:  
>[...]  
>>LSP doesn't know what a SysEnvPtr is.  
>  
>Use a SysEnvRec, not a SysEnvPtr. IM Vol V is incorrect in its

>declaration, at least as far as I can tell. Check out Tech Note 129.

I had the same problem and came up with the same 'solution'.

---

## USENET Macintosh Programmer's Guide

---

While this is just an error in IM, it would be nice if THINK Pascal included proper interface files for the stuff that's defined internally in THINK Pascal. I remember when I ran across this problem I went to look for the interface definition of the SysEnviron() call. No dice. When THINK P. defines something internally, they just have a blank file for the interface. Why not put the proper interfaces in, so that we can see exactly what THINK P. wants, and comment them out, so that it doesn't interfere with whatever mechanism is currently in place?

Robert

= gft\_robert@gsbacd.uchicago.edu

●●●

From: ccc\_ido@waikato.ac.nz (Lawrence D'Oliveiro, Waikato University)

**Subject: Alternative pkg interfaces for Pascal**

Ever found the standard Pascal definition for SGetFile/SFPGetFile to be a nuisance? I have. An SFileTypeList of 4 elements is almost never what I want. Often I just want one type. Several times I've wanted a dynamic list of types, which could be of any length.

Also, call me irresponsible, but I think it's inelegant to have an assembly- language glue routine when a short inline will do the trick.

So here my alternative declarations of some toolbox calls. First of all, the disk-initialisation package in DiskInit.p:

```
Procedure DIILoad;

    Inline
    $3F3C, $0002,      {move.w #2, -(sp)}
    $A9E9;            {_Pack2}

Procedure DIUnload;

    Inline
    $3F3C, $0004,      {move.w #4, -(sp)}
    $A9E9;            {_Pack2}

Function DIBadMount
(
    where : Point;
    evtMessage : LongInt
) : Integer;

    Inline
    $4267,      {clr.w -(sp)}
    $A9E9;      {_Pack2}

Function DIFormat
(
    drvNum : Integer
) : OSErr;

    Inline
    $3F3C, $0006,      {move.w #6, -(sp)}
    $A9E9;            {_Pack2}

Function DIVERify
(
    drvNum : Integer
) : OSErr;
```

```
Inline
$3F3C, $0008,      {move.w #8, -(sp)}
$A9E9;      {_Pack2}
```

---

## USENET Macintosh Programmer's Guide

---

```
Function DIZero
(
    drvNum : Integer;
    volName : Str255
) : OSErr;

    Inline
    $3F3C, $000A,      {move.w #10, -(sp)}
    $A9E9;             {_Pack2}
```

And here are new definitions for most of the routines in Packages.p (IUCompString and IUEqualString are a little more complicated, so I leave them as glue). SFGetFile and SFPPGetFile have their typeList arguments declared as "Ptr", so you can pass just about anything you like.

Oh, and StringToNum is a function, which I've found to be far more convenient, particularly when I only want an integer result.

```
{ Standard File Package -----}

Procedure SFPutFile
(
    where : Point;
    prompt : Str255;
    origName : Str255;
    dlgHook : ProcPtr;
    var reply : SFReply
);

    Inline
    $3F3C, $0001,      {move.w #1, -(sp)}
    $A9EA;             {_Pack3}

Procedure SFPPutFile
(
    where : Point;
    prompt : Str255;
    origName : Str255;
    dlgHook : ProcPtr;
    var reply : SFReply;
    dlgID : Integer;
    filterProc : ProcPtr
);

    Inline
    $3F3C, $0003,      {move.w #3, -(sp)}
    $A9EA;             {_Pack3}

Procedure SFGetFile
(
    where : Point;
    prompt : Str255;
    fileFilter: ProcPtr;
    numTypes : Integer;
    typeList : Ptr;
    dlgHook : ProcPtr;
    var reply : SFReply
);

    Inline
    $3F3C, $0002,      {move.w #2, -(sp)}
    $A9EA;             {_Pack3}
```

```
Procedure SFPGetFile  
(  
    where : Point;
```

---

## USENET Macintosh Programmer's Guide

---

```
prompt : Str255;
fileFilter : ProcPtr;
numTypes : Integer;
typeList : Ptr;
dlgHook : ProcPtr;
var reply : SFReply;
dlgID : Integer;
filterProc : ProcPtr
);

Inline
$3F3C, $0004,      {move.w #4, -(sp)}
$A9EA;      {_Pack3}

{ International Utilities Package -----}

Function IUGetInt1
(
theID : Integer
) : Handle;

Inline
$3F3C, $0006,      {move.w #6, -(sp)}
$A9ED;      {_Pack6}

Procedure IUSetInt1
(
refNum : Integer;
theID : Integer;
intlParam : Handle
);

Inline
$3F3C, $0008,      {move.w #8, -(sp)}
$A9ED;      {_Pack6}

Procedure IUDateString
(
dateTime : LongInt;
longFlag : DateForm;
var result : Str255
);

Inline
$4267,      {clr.w -(sp)}
$A9ED;      {_Pack6}

Procedure IUDatePString
(
dateTime : LongInt;
longFlag : DateForm;
var result : Str255;
intlParam : Handle
);

Inline
$3F3C, $000E,      {move.w #14, -(sp)}
$A9ED;      {_Pack6}

Procedure IUTimeString
(
dateTime : LongInt;
wantSeconds : Boolean;
```

---



```
var result : Str255  
);
```

---

## USENET Macintosh Programmer's Guide

---

```

    Inline
    $3F3C, $0002,      {move.w #2, -(sp)}
    $A9ED;           {_Pack6}

Procedure IUTimePString
(
    dateTime : LongInt;
    wantSeconds : Boolean;
    var result : Str255;
    intlParam : Handle
);

    Inline
    $3F3C, $0010,      {move.w #16, -(sp)}
    $A9ED;           {_Pack6}

Function IUMetric : Boolean;

    Inline
    $3F3C, $0004,      {move.w #4, -(sp)}
    $A9ED;           {_Pack6}

Function IUMagString
(
    aPtr, bPtr : Ptr;
    aLen, bLen : Integer
) : Integer;

    Inline
    $3F3C, $000A,      {move.w #10, -(sp)}
    $A9ED;           {_Pack6}

Function IUMagIDString
(
    aPtr, bPtr : Ptr;
    aLen, bLen : Integer
) : Integer;

    Inline
    $3F3C, $000C,      {move.w #12, -(sp)}
    $A9ED;           {_Pack6}

{ Binary-Decimal Conversion Package -----}

Function StringToNum
(
    theString : Str255
) : LongInt;

    Inline
    $205F,      {move.l (sp)+, a0}
    $3F3C, $0001,      {move.w #1, -(sp)}
    $A9EE,      {_Pack7}
    $2E80;      {move.l d0, (sp)}

Procedure NumToString
(
    theNum : LongInt;
    var theString : Str255
);

    Inline
    $205F,      {move.l (sp)+, a0}
```

---

```
$201F,      {move.l (sp)+, d0}  
$4267,      {clr.w -(sp)}  
$A9EE;      {_Pack7}
```

---

## USENET Macintosh Programmer's Guide

---

Lawrence D'Oliveiro

●●●

From: jackiw@cs.swarthmore.edu (Nick Jackiw)

**Subject: Re: Bug in THINK Pascal? My bug?**

czychi@bernina.ethz.ch.UUCP (Gary Czychi) writes:

> Hi there,

Hi. I tried to send mail, but my mailer choked.

```
> .
> .
> Writeln('Pick a point');
> repeat
>   GetMouse(Horiz, Vert);
> until Button;
>
> MoveTo(firstHoriz, firstVert);
> .
> .
>
> But whenever I use this loop, it seems as if every single statement until
> the end of the program or until another repeat..until loop is passed
> without execution!
```

By this do you mean that the text "Pick a point" never appears in the text window? If so, beat; it sounds like something is seriously scrambled on your disk (i. e. your project file, your THINK Pascal application, or your system folder---it's not a bug in LSP, because the exact same fragment compiles and executes fine on my system, which is identical to yours).

However, if the text \*does\* appear but the program doesn't pause to get the mouse, beware that if your button is still held down from some previous action (like a previous repeat/until button), this repeat loop will terminate immediately. When using constructs like this, it's always wise to preface them with

```
repeat
until (not button)
```

which will simply pause the program until it's certain that the user is no longer holding down the button. Then the next repeat/until button is guaranteed to cycle until the user clicks the button \*for that loop\* (rather than for some previous one).

I hope that's the problem. If not, try removing the objects from your project and rebuilding it, reinstalling THINK, and finally, reinstalling your system.

```
> The only possibility for me to achieve a correct behavior is, when I insert
> a 'stop' in the line directly after the loop and use 'go' until the end of
> the program. Then everything works as expected.
>
> I'm using THINK Pascal 3.0 on a SE/30 with 6.0.5. Maybe I've found a bug??
>
> Thanks a lot for any help.
>
> Gary T. Czychi          University of St.Gallen, Switzerland
```

Nicholas Jackiw

●●●

From: minow@mounntn.dec.com (Martin Minow)

**Subject: Re: A possible bug in THINK C 4.0.1**



---

## USENET Macintosh Programmer's Guide

---

In article <425@spot.wbst128.xerox.com>, rainero.wbst@xerox.com (EmilRainero) asks about a possible odd-address error in Think C in coderoughly organized as:

```
char    something;  
Str255 string;
```

One thing you might check is whether you're passing the address of "something" to a Toolbox variable that expects the address of a Byte or Boolean. Both of these are, suprisingly, 16-bit quantities. As a result, if "something" is at an odd-address, your program will crash in that toolbox routine.

Guess how I discovered this?

Martin Minow

•••

From: jackiw@cs.swarthmore.edu (Nick Jackiw)

**Subject: >32k arrays (in Think Pascal)**

spellbinder@uwav1.u.washington.edu writes:

```
> Here is a compiler question for those programmers out in net-land.  
>  
>   type ArrayRecord = record  
>       arraySize : longint;  
>       signal : array [ 0..1 ] of real;  
>       end;  
>   ArrayRecordPtr = ^ArrayRecord;  
>   ArrayRecordHdl = ^ArrayRecordPtr;  
>  
> The procedure knows how much data there is by getting the arraySize variable.  
>  
> Now I'm writing this code in THINK Pascal version 2.03. It works fine when the  
> amount of memory dedicated to the signal array is less than 32K, but when it  
> gets bigger than this, then things start to mess up. So basically, for array  
> sizes less than 8K data points, everything's fine. My idea is that THINK Pascal  
> has a problem getting a value at an offset greater than 32K from the start  
> of the variable. I have verified my idea by changing signal to become  
> an array of double and an array of extended and each time the procedure  
> craps out when the data exceeds 32K. My feeling is that the compiler is not  
> generating the correct code to access values beyond 32K.  
>  
> My questions are this:  
>  
>   1) Is there another explanation for the problem?  
>  
>   2) Are there any work arounds in THINK Pascal version 2.03?  
>  
>   3) Will THINK Pascal 3.01 avoid the problem?  
>  
> Thanks in advance.  
>  
> Blair Zajac           Zajac@phast.phys.washington.edu
```

Your diagnosis is correct: Pascal uses the 68K's "Address Register Indirect With Displacement" addressing mode to access array data, and this mode's displacement (the offset) is limited to a sign-extended 16-bit displacement integer. Thus, you're limited to 32K arrays.

[A side note: in that it's sign-extended, it'd be possible to access 64K worth of data, if your base address was the arrayStart+32K, I think. Not that any Pascal compiler I've seen does this, of course...]

---

## USENET Macintosh Programmer's Guide

---

The work around is the same for THINK 2 and 3--you'll have to calculate your own offset. The best way to do this is with a bit of inline assembler:

```
type RealPtr=^real;
```

---

## USENET Macintosh Programmer's Guide

---

```
function MyData(Where:ArrayRecordHdl; Index:integer):realPtr;

inline $3017, $c0fc, $0004, $206f, $0002, $2050, $d1c0, $2f48, $0006, $5c4f;
```

To access your data, where formerly you used

```
ArrayRecordHdl^[50]
```

now use

```
MyData(ArrayRecordHdl,50)^
```

Note that the third word of the in-line is \$0004: this is the hardwired size of a single element of data in byte (1 REAL=4 bytes). The code compiles as:

```
MOVE.W    (A7),D0      ; Get the index off the stack
MULU.W    #4,D0        ; Multiply by the element size (32bit result)
MOVEA.L   2(A7),A0     ; Get a handle to the array
MOVEA.L   (A0),A0      ; Dereference it
ADDA.L    D0,A0        ; Add it to the calculated offset
MOVE.L    A0,6(A7)     ; Put resulting ptr back on the stack
ADDQ.W    #6,A7        ; and throw away arguments
```

Be sure to change \$0004 to SizeOf(elem) if your elements ever change from being real. This code generates is a bit less efficient than Pascal's native array indexing, but it gets you past the 32K problem. Note the complete lack of range checking...hope it's okay.

---

jackiw@cs.swarthmore.edu

...

From: chewy@apple.com (Paul Snively)

**Subject: Re: Multiple Inheritance for HandleObjects in C++**

In article <15132@reed.UUCP> bowman@reed.UUCP (Eric Bowman) writes:

```
> I've just discovered, to my horror, that you can't create HandleObjects
> with multiple base classes in MPW C++. Are there any work arounds for this?
> Anyone know why this is the case?
```

Larry Rosenstein and Amanda Walker have already done an excellent job of answering the question Eric poses, but I'd like to point out something else:

Since Eric is using HandleObjects, he apparently isn't concerned about Object Pascal compatibility. This means that he's probably concerned about fragmenting the heap, which standard C++ objects, being based on malloc and free, tend to do.

Andy Shebanow, a former engineer here in MacDTS, wrote a very good article in "D e v e l o p" describing a class that he created called "PtrObject." The idea was to provide all of the functionality of standard C++ but to use NewPtr and DisposPtr to manage memory instead of malloc and free. Perhaps Eric would benefit from using Andy's PtrObject class.

Just a thought...

Paul Snively

...