HeapTrace(tm) v2.0

Heap Memory Debugging for Borland Delphi(tm) Programs

73000.1064@compuserve.com

At last!

License and Copyright

The software product "HeapTrace" and any accompanying written materials (to be hereinafter collectively referred to as the "Software"), is and shall remain the property of Modelistica, which reserves all rights to reproduce, copy or distribute the Software, except as otherwise provided for in this agreement. The Software is protected by all applicable copyright laws and international treaties.

If you intend to use the Software on a regular basis, you must register your copy of the Software with Modelistica. See the Registration section of this document for details. As a registered user of the Software, Modelistica will grant you a personal right to use the Software and you will be eligible for technical support and upgrade notifications.

If you use this product
You will remain solely responsible to anyone receiving a program made by you which includes any part of this Software. You will indemnify and hold Modelistica and its suppliers harmless from and against any claims or liabilities arising out of the use, reproduction or distribution of the program.

This program is provided "as is" without warranty of any kind, either expressed or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose.

In no event will Modelistica be liable to you for any damages, including, but not limited to, any lost profits, lost savings, or any other incidental or consequential damages arising out of the use or inability to use this program, even if Modelistica has been advised of such damages, or for any other claim by any other party.

If for any reason any portions of this agreement are not enforceable or unlawful then all other remaining portions will still form a valid license agreement.

Distribution
You may distribute as many copies of the shareware versions of HeapTrace and HeapTrace32 as long as the distribution files and the installation executable are kept unaltered. It is unlawful to charge or to request donations or fees for the distribution of HeapTrace without express authorization from Modelistica.

Installation

HeapTrace is distributed as a self extracting self installing file called HEAPTRAC.EXE. The distribution file for HeapTrace32 is HT32.EXE. To install HeapTrace:

- Run HEAPTRAC.EXE (or HT32.EXE for HeapTrace32)
- Choose an installation directory when prompted.
- Include the chosen directory in Delphi's Options/Project/Directories/Search_Path menu option.
- Point your PATH variable to the chosen directory, or move HT16.DLL or HT32.DLL to a directory belonging to you program's search path.

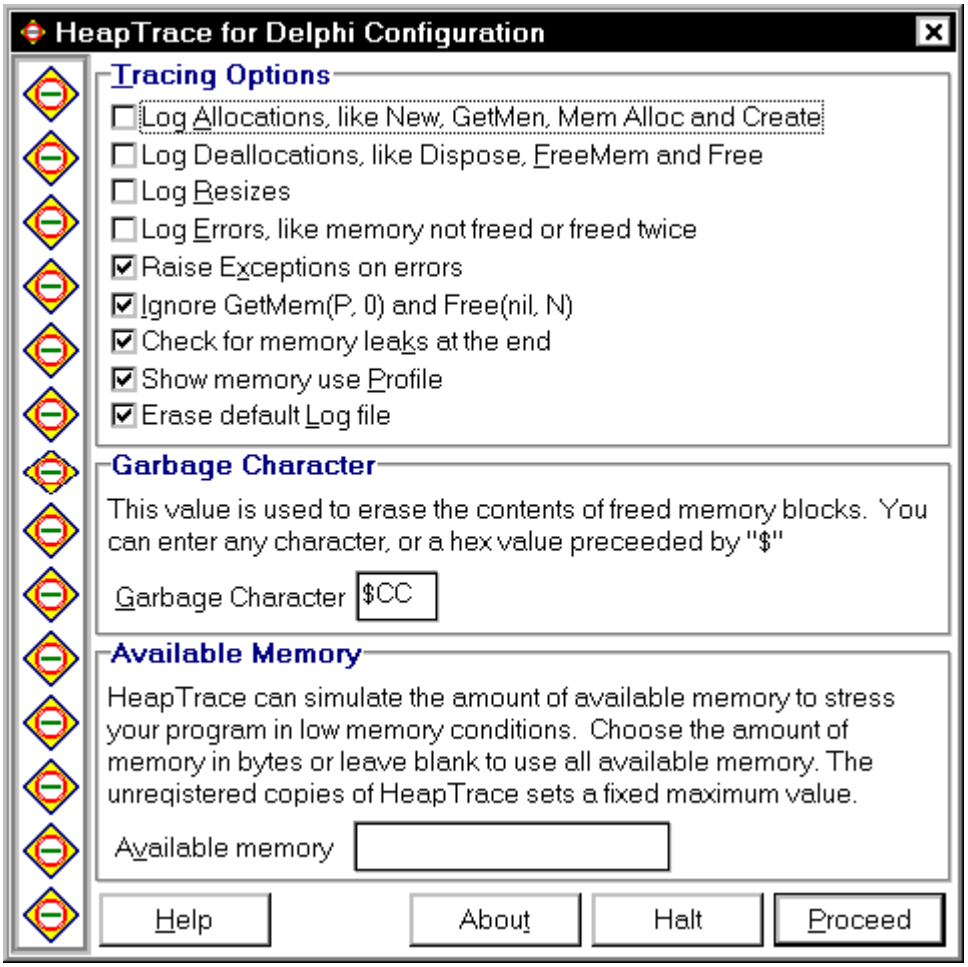
You can now load the sample program TestHT16.DPR or TestHT32.DPR into Delphi to take a look at what HeapTrace and do.

Where to Start

Using HeapTrace in your programs is easy. Just include the **HEAPTRAC** unit as the first unit in your main programs USE clause, like this:

```
program MyApp;
{ file MYAPP.DPR }
uses
    HeapTrac, { ** like this **}
    { other units used by your main program }
    Forms;
{  the rest of your main module }
end.
```

When you want HeapTrace activated, just add a **/HEAPTRACE** switch to your program, either through the Run/Paremeters menu option in Delphi, or in any command line you use to launch your program. When activated, HeapTrace will pop up a configuration dialog. The options you select will affect the behavior of HeapTrace. Once you set the desired options, you can press the button labeled "Proceed" to start tracing your application . HeapTrace will save the selected options in the file HEAPTRAC.IN I in the same directory as your application, and use them as default the next time you run the program. If you press the "Halt" button, HeapTrace will stop your program immediately.



If you turn on any of the logging options, HeapTrace will create a file called **MYAPP.HLG** containing the details of the heap memory activity performed by your program. The file can also contain entries for the heap related errors that HeapTrace detected. Each entry will specify the time at which the event occurred, the address in your program that generated the event and other useful information. Following is a sample of a HeapTrace log file. The first address in each line, after de equals sign (=) is the memory block's address. The number between square

brackets ([ ])is the block's size. The lines with messages ending in two exclamation marks (!!) indicate dynamic memory errors. Some question marks (?) may appear in place of program addresses with HeapTrace32. They indicate untraceable parts of your program.

```
[1996/02/15 18:01:37]
 1=$50D7:0B78 [     3] Allocated        at $0009:0BFA
 2=$50D7:002C [    12] Allocated        at $0000:0000 class TList
 3=$50D7:005C [    26] Allocated        at $0001:00DB class TComponent
 4=$50D7:0098 [    28] Allocated        at $0001:00F2 class TStringList
 5=$50D7:00D8 [    12] Allocated        at $0001:00F2 class TList
 6=$50D7:002C [    12] Freed            at $0001:2CC1 class TList
 7=$50D7:002C [     ?] Free Invalid  !! at $0001:010F
 8=$50E7:0020 [ 12345] Allocated        at $0001:0117
 9=$501F:0020 [ 12345] Allocated        at $0001:0126
10=$108F:0020 [  4321] Allocated        at $0001:0135
11=$501F:0020 [ 12345] Overrun       !! at $0001:015C
12=$501F:0020 [     ?] Free Invalid  !! at $0001:015C
13=$501F:0020 [     ?] Free Invalid  !! at $0001:016C
14=$108F:0020 [  4321] Free Invalid  !! at $0001:017C
15=$0000:0000 [ 12345] Out of Memory !! at $0001:01A2
15=$50E7:0020 [ 12345] Not Freed     !! at $0001:0117
16=$50D7:00D8 [    12] Not Freed     !! at $0001:00F2 class TList
17=$50D7:0098 [    28] Not Freed     !! at $0001:00F2 class TStringList
18=$50D7:005C [    26] Not Freed     !! at $0001:00DB class TComponent
19=$50D7:0B78 [     3] Not Freed     !! at $        ?
```

That's it! You can start tracing now, but do come back for some advanced information. For more information on tracing options see the Choosing What to Trace and Events Traced sections of this manual.

The shareware version of HeapTrace sets a limited amount of memory for your program.

Note: Delphi 1.x Smart Callbacks must be used with HeapTrace 16bit for it to work correctly. In the IDE, select Options/Project/Compiler and check the Smart callbacks box.

Choosing What to Trace

For non trivial programs, logging all trace activity from program start to finish might be unnecessary and can consume a lot of disk space. HeapTrace let's you select what gets traced and what not, by using the SetHeapTraceOptions function to set any group of the following flags:

```
type
  { tracing options }
  THeapTraceOption = (
      htoLogAlloc,           { log allocations     }
      htoLogFree,            { log deallocations   }
      htoLogResize,          { log resizes         }
      htoLogErrors,          { log errors          }
      htoRaiseOnErrors,      { generate exceptions }
      htoIgnoreZeroNil,      { ignored GetMem(p, 0)
                               and FreeMem(nil, N) errors }
      htoProfiling,          { create an in memory log }
      htoCheckForLeaks,      { check for unfreed blocks at shutdown  }
      htoConfigDialog,       { launch configuration dialog at startup }
      htoProfileDialog,      { launch memory profile dialog at shutdown }
      htoEraseOldLog         { erase old log files }
    );
  THeapTraceOptions = set of THeapTraceOption;

const
  htoFullTrace = [htoLogAlloc..htoLogErrors];
```

These options are also available from the configuration dialog that pops up when you use HeapTrace in your programs, or through the command line switches.

The option htoRaiseOnErrors tells HeapTrace to raise an exception as soon as any error is detected, so you get informed of the error right away.

The other options tell HeapTrace what kind of events should be sent to the logging routine. Every time a block of memory is allocated, HeapTrace examines the current set of options and associates it with the block. The block will "remember" the options that were in effect then on, so the same options will apply to any operation on the memory block even when the global set of options has been changed with SetHeapTraceOptions.

Introduction and Key Features

HeapTrace enables debugging of heap memory use in Borland Delphi(tm) applications. HeapTrace helps you get rid of memory leaks, dangling pointers and memory overruns in your programs. It provides optional logging of all memory allocations, de-allocations and errors.

HeapTrace is extremely easy to use. Just include "HeapTrac" as the first unit in your main program's (.DPR) "USE" clause. HeapTrace is so fast, that you will hardly notice it's presence once your program has loaded. This version of HeapTrace is complete, functional and not nagged or crippled.

This document the user manual for HeapTrace. There are versions of HeapTrace for both Delphi 1.x and for 32bit Delphi2.x. This manual covers both. HeapTrace is so easy to use, that you may be tempted to skip this manual altogether, but, before you do, please take a look at the license and registration information.

HeapTrace's key features and installation are covered in the other sections.

Future Releases

There are new releases of HeapTrace planned in the future. Registered users of heap trace will be notified of upgrades when available. Interim versions and bug correction releases will be delivered to registered users for free.

- In a future release, Delphi's "**is**" and "**as**" operators will be substituted at runtime by versions that don't produce general protection faults and that will pass any errors through the logging routines.
- HeapTrace will provide.graphs that show memory consumption statistics throughout a run of your program.
- Future versions will let you examine the list of allocated memory blocks at run time, as well as change configuration options on-the-fly.

Technical Notes and Caveats

- HeapTrace 16bit needs that Delphi 1.x Smart Callbacks are used. You can turn on Smart Callbacks in the IDE by selecting the Options/Project/Compiler menu option.
- HeapTrace uses around 32 bytes of memory for each memory allocation. This means that a 16 bit Delphi 1.x program will not be able to allocate a block larger than 65500 or so bytes while being traced.
- To be able to search for the error addresses reported by HeapTrace you have to turn debug information on in Delphi's compiler options.
- Due to the kind of optimizations performed by Delphi 2.0, HeapTrace32 cannot always track the program address that produced the heap activity. Turning off optimizations and turning on stack frames lets HeapTrace32 hit most addresses. See the HeapTrace32 section for details.
- The error address that appears on "Not Freed" messages is the address where the memory block was allocated. This gives a good idea of what part of the program is failing to release the memory.
- You should strive to remove all known bugs from your programs before using HeapTrace. A buggy program will produce many heap related error messages even when the errors are originally unrelated to heap memory management.
- HeapTrace does a fair amount of patching over the Delphi Run Time Library (RTL). To improve performance, HeapTrace leaves the patched parts of the RTL exposed to subsequent writes. A "wild pointer" in your application could effectively trash the RTL if it touches any of the exposed parts.

Registering Heap Trace

HeapTrace is a copyrighted product that is published as shareware. That means that if you intend to use HeapTrace on a regular basis you should register.

16bit and 32bit versions

The 16bit and the 32bit versions of HeapTrace are different products. That means that you should register each version separately. Please contact Modelistica through e-mail at 73000.1064@compuserve.com for HeapTrace32 details and availability.

Distribution

HeapTrace is distributed as a self extracting and self installing executable called HEAPTRAC.EXE. The distribution file contains a complete copy of the fully functional product. If you are reading this, you should already have the complete product in your computer.

Pricing

The registration price for each of HeapTrace 2.0 and HeapTrace32 2.0 is just US$40.. Students may use any version of HeapTrace without any charge as long as their use is for academic work.

What you get When you Register

Upon registration, Modelistica will send you an End User's License Agreement. You will also be notified of new releases and will have access to technical support via e-mail.

Limited Time Offer!

The HeapTrace introductory offer has been extended. Users who register either version of HeapTrace via CompuServe before April 15, 1996 will also receive a license for the matching 16bit or 32bit version at absolutely no charge. So hurry up and register now.

How to Register

The easiest way to register is by using CompuServe. To register via CompuServe, **GO SWREG** and select **registration Id 10866**. This method of registration is by far the fastest way to get your user's license, information about new versions of HeapTrace and to receive your free copy of HeapTrace32 or HeapTrace16 when applicable.

You can also register by sending a check payable to Juancarlo Anez to the following address:

> Juancarlo Anez
> Modelistica
> AP 47709
> Caracas, 1041A
> Venezuela

If you choose to register by postal mail, you must still provide a an e-mail address to receive your license agreement. Users who register through postal mail do not qualify for the free HeapTrace32 offer.

VCL Memory Leaks

In both Delphi 1.x and Delphi 2.0 there are some of the units in the VCL library allocate several memory blocks on initialization, but fail to release them on application shut down. You can click here for a partial list of VCL memory related errors. If you don't want VCL memory leaks reported, disable all options in the HeapTrace configuration dialog and change your main program module to read like this:

```
program MyApp;
{ file MYAPP.DPR }
uses
    HeapTrac,
    {...}
    Forms;
{ the rest of your declarations }
begin
    SetHeapTraceOptions(htoFullTrace) { your choice of tracing options }
{ the rest of your main module }
end.
```

Events Traced

These are the events that HeapTrace can report:

```
type
   THeapTraceEvent = (
      hteAllocated,
      hteFreed,
      hteResized,

      hteFreeInvalid,
      hteOverrun,
      hteInvalid,
      hteNotFreed,
      hteFreeNil,
      hteInvalidSize,
      hteSizeIsZero,
      hteOutOfMemory,
      hteFakeOutOfMemory
   );
```

The first three events are normal heap operations. The other events belong to program errors. Events are sent to the LogAlloc and LogMem procedural variable routines in the HeapTrac unit. You can do custom processing depending on the type of event by installing your own procedure in LogMem. See the file HEAPTRAC.INT for further documentation.

Redirecting HeapTrace Output

HeapTrace uses a Log procedure to output trace messages. You can call Log directly to have your own messages inserted in the log file.

The HEAPTRAC unit declares Log as a procedural variable, like this:

```
Log       :procedure(const Msg :string)
           = HeapTraceDefaultLog;
```

You can redirect HeapTrace's output by assigning a different procedure to the Log variable. The following unit redirects the output to a WinCrt console window:

```
unit Ht_Con;
{
// Redirect HeapTrace's output to a WinCrt console window
}
interface
implementation
uses
  HeapTrac,
  WinCrt;

procedure CrtLog(const msg :string); far;
begin
  WriteLn(msg)
end;

begin
  HeapTrac.Log := CrtLog
end.
```

HeapTrace also lets you perform specific actions depending on the type of event being logged. See the Events Traced section or the HEAPTRAC.INT file for more information.

Key Features

- **Fast, fast, fast!**
  HeapTrace is optimized so there's only a small impact on performance even on large applications.
- **Only limited by available memory**
  HeapTrace does not use any fixed-size data structures, so you can keep tracing as long as there's memory available.
- **Configurable. Make it do what you want.**
  HeapTrace lets you change the format and destination of logging output, choose what to trace, and much more.
- **Log errors or have them raise exceptions**
  You can tell HeapTrace to output heap memory errors to a log file, or have it raise an exception when the error occurs, so you can fin errors on the spot.
- **Trace each allocation and deallocation**
  Tell HeapTrace to log everything, and find out where each block of memory is being created and freed.
- **Embedable**
  You can leave HeapTrace compiled to your production executables. It will only activate itself when it sees the /HEAPTRACE option in the command line.
- **Simulate Out of Memory conditions**
  Stress your application. Tell HeapTrace what the amount of available memory should be, and let it simulate out of memory exceptions when memory runs out.
- **Specify Shutdown Actions**
  HeapTrace lets you specify actions to take place when it finishes tracing heap memory usage.
- **Memory Use Profiling**
  Find out how your program uses memory on a class by class basis.

Additional Goodies

The HEAPTRAC.INT unit defines some outstandingly useful routines that are used throughout HeapTrace. These routines provide functionality already available in Delphi but with a subtle difference: they don't produce General Protection Faults or Access Violations, ever! Here are the declarations for some of those routines. I hope that the routine names are meaningful, because they are not documented any further.

```
{ a safe way to test for an object's type }
function SafeIsOper(P :Pointer; C : TClass) : Boolean;
function SafeCast(P : Pointer; C : TClass) : Pointer;

function SafeClassType(P :Pointer)   :TClass;
function SafeClassParent(C :TClass)  :TClass;
function SafeInstanceSize(P :Pointer):TMemSize;
```

HeapTrace32 for Delphi 2.0

HeapTrace32 lets you debug memory allocation in 32bit Delphi 2.0 applications. It provides the same functionality as the 16bit version of HeapTrace.

Delphi 2.0 performs some thorough optimizations in your code. When optimizations are turned on, and stack frames are turned off, HeapTrace32 cannot always track the program address where heap activity occurred. When this happens, HeapTrace32 will show a question mark in place of the address in the MYAPP.HLG log file.

```
2=$008A8118 [    32] Allocated        at $       ? class TComponent
```

If you want HeapTrace32 to give you the correct address for every memory activity, follow these steps:

- Select Project/Options/Compiler. Turn off Optimizations and turn on Stack Frames.
- Select Project/Options/Directories&Conditionals, and in the Search Path box, add the full path to the Delphi 2.0 VCL source. If you don't have the VCL source, just skip this step. HeapTrace32 will try to track most of the addresses.
- Add HeapTrac to the uses clause of your main programs file.
- Select Project/Build All, and run your program.

What's New

This is the change history for HeapTrace in reverse chronological order.

v2.0.5

- Now the profile dialog uses the default Windows colors. The previous choice of colors interfered with some user choices..
- The name for the log file now uses the extension **.HLG**. The .LOG extension conflicted with files created by other tools.
- All of HeapTrace options can now be set in the command line.

v2.0.4

- New format for the registration files. Older versions of HeapTrace crash in the presence of the new .LIC files.

v2.0

- Added memory use profiling and and a memory use profile dialog.
- Displaying the configuration dialog or the memory use profile dialog and checking for memory leaks at the end are all now optional.
- All of HeapTrace options can now be set in the command line.

v1.2.2

- Fixed bug that made allocations not show up in the log file.
- Switched to Nico Mak's WinZipSE for distribution and installation. The new installation executable has about half the footprint, it's faster and has the convenience of a .ZIP file.
- The 16 bit version of HeapTrace correctly detects blocks that are too large to trace. Not an issue in HeapTrace32.
- The configuration dialog was revamped. It is now resizeable. Source code is no longer provided.

v1.2.1

- Fixed bug that made allocations not show up in the log file.

v1.2

- HeapTrace32 for Delphi 2.0 is available with this release.
- HeapTrace is now embeddable. HeapTrace will not activate itself unless it finds the /HEAPTRACE. command line option when you execute your program. This feature lets you release your programs with HeapTrace embedded, and test for memory errors at your client's site.
- HeapTrace is now compatible with some undocumented behavior in Delphi, like allocation of zero sized blocks, which returns nil, and deallocation of nil pointers, which does nothing..

v1.1

- Skipped.

v1.0.1

- Removed the limitation on memory block size. HeapTrace can handle blocks as large as 65494 bytes in size. HeapTrace32 can handle memory sizes up to MaxLongint bytes (2 megabytes).
- Added shutdown procedure chain. Client code can specify procedures to be executed when HeapTrace shuts down and removes itself from memory. See Shutdown Actions for details.

v1.0.0

- Initial release.

Shutdown Actions

HeapTrace lets client code specify actions to take place after HeapTrace shuts down and memory management has returned to normal operation. Shutdown procedures are useful when you need to perform actions that you don't want traced on application exit. Shutdown procedures are executed in reverse order: the procedure added last is executed first.

To specify a shutdown action, just call the HeapTraceOnShutdown procedure as declared in the HEAPTRAC.INT unit. Here's an example taken from the TestHTXX sample program.

```
program HT_Test;
uses
  HeapTrac, { ** },
`
..{ ** other used units }
  SysUtils;

{ shut down procedures must be declared far }
procedure ShowLogFile; far;
var
   buf :array[0..255] of Char;
begin
   WinExec(StrPCopy(
               buf,
               'NOTEPAD '+HTDefaultLogFileName),
           sw_Show);
end;

begin
   { on shut down routines are called after
     HeapTrace has been removed from the system }
   HeapTraceOnShutDownDo(ShowLogFile);

   { *** rest of program *** }
end.
```

List of VCL 1.x Memory Leaks

This is a partial list of memory leaks that HeapTrace found in VCL 1.x. Many of them are still present in Delphi 2.0.

CLASSES.PAS line 3912
Objects allocated here never freed

GRAPHICS.PAS line 4591
Objects allocated in InitGraphics are never freed.

MENUS.PAS line 1366
Like above

FORMS.PAS line 3437
FIcon is never freed

DIALOGS.PAS line 1862
Call to FreeMem with nil pointer (Delphi ignores this error, but it's undocumented behavior)

DB.PAS line 2616
GetMem with zero size (Delphi ignores this error, but it's undocumented behavior)

DB.PAS line 2637
Call to FreeMem with nil pointer (Delphi ignores this error, but it's undocumented behavior)

DB.PAS line 4536
Session never freed

DBCTRLS.PAS line 2277
FHints never freed

DBGRIDS.PAS line 4541
FTitleFont never freed

PRINTERS line ?
FTitle never freed by TPrinter.Destroy (this bug contributed by Scott Samet)

HeapTrace API

```
{
// HeapTrace
//   Dynamic memory debugging for Borland Delphi.
//
// © 1996, Modelistica, Caracas. All rights reserved
//   73000.1064@compuserve.com
}
unit HeapTrac;
{ Tracing module }
interface
uses
  SysUtils;

{$i HTCRight.pas }

type
  TMemSize  = Longint;
  PMemSize  = ^TMemSize;
  TMemTime  = Longint;    { miliseconds }

type
  { tracing options }
  THeapTraceOption = (
      htoLogAlloc,           { log allocations     }
      htoLogFree,            { log deallocations   }
      htoLogResize,          { log resizes         }
      htoLogErrors,          { log errors          }
      htoRaiseOnErrors,      { generate exceptions }
      htoIgnoreZeroNil,      { ignored GetMem(p, 0) and FreeMem(nil, N) errors }
      htoProfiling,          { create an in memory log }
      htoCheckForLeaks,    { check for unfreed blocks at shutdown  }
      htoConfigDialog,       { launch configuration dialog at startup }
      htoProfileDialog,      { launch memory profile dialog at shutdown }
      htoEraseOldLog
    );
  THeapTraceOptions = set of THeapTraceOption;

const
  htoFullTrace = [htoLogAlloc..htoLogErrors];

{ these are the events that HeapTrace watches }
type
  THeapTraceEvent = (
      hteInternalError,
      hteAllocated,
      hteFreed,
      hteResized,
      hteFreeInvalid,
      hteOverrun,
      hteInvalid,
      hteNotFreed,
      hteFreeNil,
      hteInvalidSize,
      hteSizeIsZero,
      hteOutOfMemory,
      hteFakeOutOfMemory
  );
  THeapTraceRaisableEvent = hteFreeInvalid..hteInvalidSize;

const
```

```
  HeapTraceRaisableEvents =
[Low(THeapTraceRaisableEvent)..High(THeapTraceRaisableEvent)];

const
  HTEventText : array[THeapTraceEvent] of PChar = (
      {hteInternalError}    'Errror        !!',
      {hteAllocated}        'Allocated       ',
      {hteFreed}            'Freed           ',
      {hteResized}          'Resized         ',
      {hteFreeInvalid}      'Free Invalid  !!',
      {hteOverrun}          'Overrun       !!',
      {hteInvalid}          'Invalid       !!',
      {hteNotFreed}         'Not Freed     !!',
      {hteFreeNil}          'Free Nil      !!',
      {hteInvalidSize}      'Invalid Size  !!',
      {hteSizeIsZero}       'Size is Zero  !!',
      {hteOutOfMemory}      'Out of Memory !!',
      {hteFakeOutOfMemory} 'Fake No Memory!!'
  );

type
  TAllocation = class
  public
     function  MemPtr       :Pointer;           virtual; export;
     function  MemSize      :TMemSize;          virtual; export;
     function  WhereCreated :Pointer;           virtual; export;
     function  WhenCreated  :TMemTime;         virtual; export;
     function  WhereFreed   :Pointer;           virtual; export;
     function  WhenFreed    :TMemTime;         virtual; export;
     function  LifeTime     :TMemTime;         virtual; export;
     function  Valid        :Boolean;           virtual; export;
     function  ObjectClass  :TClass;            virtual; export;
     function  Options      :THeapTraceOptions; virtual; export;
     function  Freed        :Boolean;           virtual; export;

     function  Next         :TAllocation;       virtual; export;
  end;

function HeapTraceIniFileName :string;

type
  { exceptions raised by this HeapTrace are never freed }
  EHeapTrace            = class(Exception)
    constructor Create;
    destructor Destroy;      override; { do nothing }
    destructor Kill;                   { real destructor }
    procedure FreeInstance;  override;
  end;

  { these are exceptions raised by HeapTrace }
  EHeapTraceInvalid        = class(EHeapTrace);
  EHeapTraceNil            = class(EHeapTrace);
  EHeapTraceInvalidSize    = class(EHeapTrace);
  EHeapTraceFreeInvalid    = class(EHeapTrace);
  EHeapTraceNotFreed       = class(EHeapTrace);
  EHeapTraceOverrun        = class(EHeapTrace);
  EHeapTraceSizeIsZero     = class(EHeapTrace);
  EHeapTraceTooLarge       = class(EHeapTrace);
  EHeapTraceInternalError  = class(EHeapTrace);
  EHeapTraceInvalidLogProc = class(EHeapTrace);
  EHTAllocationChainTooLong= class(EHeapTrace);

  { raised when HeapTrace simulates an Out of Memory condition }
```

```
  EHTSimulatedOutOfMemory = class(EOutOfMemory);

{ setting/getting options }
function  HeapTraceOptions :THeapTraceOptions;
procedure SetHeapTraceOptions(Value :THeapTraceOptions);
procedure ChangeHeapTraceOptions(Value :THeapTraceOptions; Enable :Boolean);

{ logging }
function  HeapTraceDefaultLogFileName :string;
procedure HTDefaultLog(const Msg :string);
procedure HTDefaultLogMem(Mem :Pointer; Size :TMemSize; C :TClass; Addr :Pointer;
Event :THeapTraceEvent);
procedure HTDefaultLogAlloc(a :TAllocation; Addr :Pointer; Event :THeapTraceEvent);

const
   { Log is used by LogMem for final output }
   Log      :procedure(const Msg :string)
             = HTDefaultLog;
   LogMem   :procedure(Mem :Pointer; Size :TMemSize; C :TClass; Addr :Pointer; Event
:THeapTraceEvent)
             = HTDefaultLogMem;
   { by default, LogAlloc directs it's output to LogMem }
   LogAlloc :procedure(a :TAllocation; Addr :Pointer; Event :THeapTraceEvent)
             = HTDefaultLogAlloc;

{ traced-heap information routines }
function  TracedMemSize(Mem :Pointer):TMemSize;
function  TracedMemoryValid(Mem :Pointer; Size :TMemSize):Boolean;

{ simluating out of memory conditions }
function  HeapTraceAvailableMemory :TMemSize;
procedure SetHeapTraceAvailableMemory(NewSize :TMemSize);
function  HeapTraceAllocatedMemory    :TMemSize;
function  HeapTraceMaxAllocatedMemory :TMemSize;

{ the garbage character is used to erase unused memory }
function  HeapTraceGarbageChar :Char;
procedure SetHeapTraceGarbageChar(Value :Char);

{ hook to perform actions after HeapTrace has shut down }
type
  THTShutDownProc = procedure;

  procedure HeapTraceOnShutdownDo(Proc: THTShutDownProc);

{ check the heap for blocks that were not freed }
  function  HeapTraceAllocationChain :TAllocation;
  procedure HeapTraceCheckForMemoryLeaks;
  procedure HeapTraceClearAllocationChain;

{ The declarations here on should not be of interest to
  most users.  They are low level stuff useful only to
  developers building enhancements to HeapTrace }

function HeapTraceActive :Boolean;

{ a safe way to test for an object's type }
function SafeIsOper(P :Pointer; C : TClass) : Boolean;
function SafeCast(P : Pointer; C : TClass) : Pointer;

function SafeClassType(P :Pointer)   :TClass;
function SafeClassParent(C :TClass)  :TClass;
function SafeInstanceSize(P :Pointer):TMemSize;
```

```
function SafeConvertAddr(Address: Pointer): Pointer;
function ConvertExceutableAddr(Addr :Pointer):Pointer;

function CallerAddr        :Pointer;
function CallersCallerAddr :Pointer;
function CallerAtOffset(Offset :Integer) :Pointer;
function AddressAtIPOffset(Offset :Integer):Pointer;

function MemSizeToStr(Size :Double):string;
function StrToMemSize(s:string):TMemSize;

type
  PtrToLong = Longint;
  LongToPtr = Pointer;
  PPointer = ^Pointer;

  TMemEvent   = procedure (Mem :Pointer; Size :TMemSize; C :TClass; Addr :Pointer;
Event :THeapTraceEvent);
  TAllocEvent = procedure (a :TAllocation; Addr :Pointer; Event :THeapTraceEvent);

  function VerMinor :Integer;
  function VerMajor :Integer;
  function VerDot   :Integer;
  function VerName  :PChar;

implementation
end.
```

Simulating Out of Memory Conditions

HeapTrace lets you stress your programs by simulating out of memory conditions. When your program runs out of simulated memory, HeapTrace will raise a EFakeOutOfMemory exception, which descends directly from Delphi's EOutOfMemory.

To set the amount of memory available to your program, use the configuration dialog or call the following routine in selected points on your program.

```
SetHeapTraceAvailableMemory(AmountOfMemory);
```

Calling the routine with a parameter of zero, stops the simulation.

Shareware vs Registered

The following is a summary of the differences between the shareware and registered versions of HeapTrace.

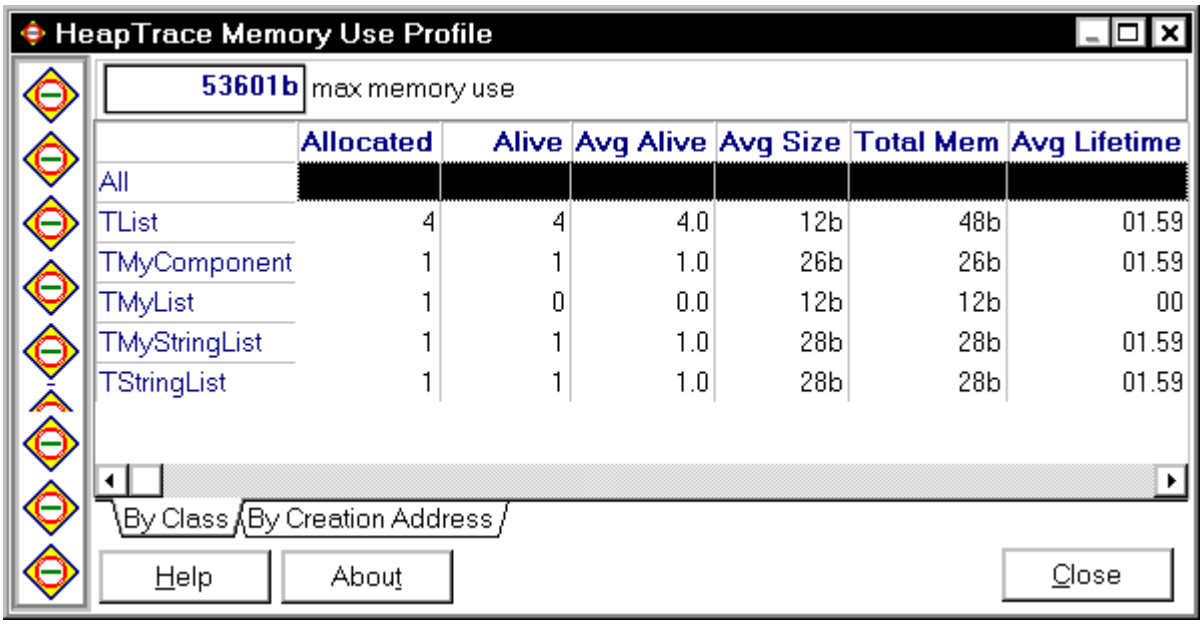
- The following options are turned on by default in the shareware version and cannot be disabled.

```
'P': htoProfiling
'K': htoCheckForLeaks
'D': htoProfileDialog
'L': htoEraseOldLog
```

- The About dialog box always pops up when the shareware version of HeapTrace is used.
- The shareware version simulates a limited amount of memory. The amount of memory can be decreased with SetHeapTraceAvailableMemory, but it cannot be increased.

Memory Use Profile

By default, HeapTrace performs some intensive profiling of your programs memory use. If "Show memory use Profile" is selected in the HeapTrace configuration dialog, or the htoProfileDialog option is set through SetHeapTraceOptions prior to program termination, HeapTrace will show the following dialog before exiting.



At the top, you'll find your programs maximum memory consumption. On the grid below, you'll find some statistics of overall and class-by-class memory usage.

Command Line Switches

All the options accessible through the SetHeapTraceOptions routine can be enabled through the command line by following the **/HEAPTRACE** command line switch with a colon and a sequence of letters and symbols indicating which options to set or remove. The syntax is:

```
/HEAPTRACE:{O[+-]}
```

For example, **/HEAPTRACE:XD** will raise exceptions on errors and display the memory use profile dialog at the end. The recognized options and their meanings are the following:

```
'A': htoLogAlloc
'F': htoLogFree
'R': htoLogResize
'E': htoLogErrors
'X': htoRaiseOnErrors
'I': htoIgnoreZeroNil
'C': htoConfigDialog
'P': htoProfiling
'K': htoCheckForLeaks
'D': htoProfileDialog
'L': htoEraseOldLog
```

If you don't specify any command line switches, HeapTrace will present the configuration dialog before letting the program start. Remember also that the last four options cannot be turned off in the shareware version of the software.

User's Manual