NS-DebugMem

NS-DebugMem lets you search for problems linked to memory use in NS-DK, NatStar and NatWeb projects.

Memory problems are detected independently of their impact on execution. It is then possible to correct stability problems simply.

You will find in this chapter

- The various NS-DebugMem functionalities.
- How to use DEBUGMEM and MEMTRACE modes.
- How to use the SPYMEM memory monitor.
- How to look up memory allocation.

Contents

Introduction

NS-DebugMem provides a series of functionalities that let you investigate problems linked to memory use in NS-DK, NatStar and NatWeb projects.

This set of files provides you with five different modes of investigation:

- ‘**DEBUG**' mode: runs an application and generates a trace file giving details of memory allocation that has not been freed or memory corruption.
- ‘**CHECK**’ mode: checks the validity of NCL's MOV and FILL instructions in the allocated memory.
- ‘**SPY**’ mode: displays the amount of memory allocated in real time via a specialized window (Monitor).
- **‘TRACE with Recompilation’** mode: runs an application and generates a trace file giving details of memory allocation that has not been freed or memory corruption with information that lets you find the corresponding lines of NCL code easily.
- ‘**API**’ mode: from an NCL program, looks up the various quantities of memory allocated using the GETMEMORYINFO% API, and checks the integrity of memory blocks allocated using the CHECKMEMORY API.

Trace File

On exiting the program, a trace file is created in the directory indicated in the NS-TRACE environment variable (or NS_TRACE under UNIX). It contains all the trace data.

⚠ The named or shared memory is not traced.

Activating DEBUGMEM mode without recompiling

Unfreed trace mode

Runs programs with the NSDBGMEM environment variable with the value DEBUGMEM.

Example :

```
local POINTER l_hseg%

new 1000, l_hseg%, "ETIQUETTE"

ST_ADRESSE = l_hseg%
FILL l_hseg%, 1000, 0
fill l_hseg%, 3, 65

setdata l_hseg%
```

The trace file contains a list of blocks allocated but not freed.

Example of the .LOG file generated:

```
Program file mame  : tstmemlb.exe
program start Time : 15:19:46

Application memory leak
Unfreed blocks at application ending
------------------------------------
Size=    44 value=10064D72
Size= 20000 value=1005FF0E
Size=    33 value=1005FEAA
Size=    11 value=1005FE5E
```

Allocation / Unfreed trace mode

Runs programs with the NSDBGMEM environment variable with the value DEBUGMEMALL.

Example :

```
local POINTER l_hseg%

new 1000, l_hseg%, "ETIQUETTE"

ST_ADRESSE = l_hseg%
FILL l_hseg%, 1000, 0
fill l_hseg%, 3, 65

setdata l_hseg%
```

The trace file contains a list of all the blocks allocated with their order number as well as a list of blocks allocated but not freed.

Example of the .LOG file generated:

```
Program file mame  : tstmemlb.exe
```

```
program start Time : 15:44:25

Allocated Blocks
================
Number =  2229 Size=    11 value=1005FE5E
Number =  2230 Size=    22 value=1005FEAA
Number =  2231 Size= 20000 value=1005FF02
Number =  2232 Size=    44 value=10064D66
Number =  2233 Size=    33 value=10064DD6

Application memory leak
Unfreed blocks at application ending
------------------------------------
Number=  2232 Size=    44 value=10064D66
Number=  2231 Size= 20000 value=1005FF02
Number=  2230 Size=    22 value=1005FEAA
```

FILL and MOV instruction systematic verification mode

This mode lets you test the validity of the MOV, FILL and most string processes in the allocated memory and alert the user concerning any problems detected.

Use

Set the environment variable NSDBGMEM to the following values which can be mixed, separated by a semi-colon.

- CHECKFILL
- CHECKMOV
- CHECKSTRINGS
- CHECKALL
- ADVERTUSER
- ONERRORTRUNCLEN
- ONERRORDOBADACTION
- ONERRORABORT

CHECKFILL: Checks for overflows with the FILL function in the allocated memory

Runs programs with the NSDBGMEM environment variable with the value CHECKFILL

The trace file contains a list of blocks in which the program has tried to execute a FILL and where there has been an overflow.

Example of the .LOG file generated:

```
Command  : wFill
Error    : length invalid, required=50, max allowed=44
Action   : Action done
```

CHECKMOV: Checks for overflows with the MOV function in the allocated memory

Runs programs with the NSDBGMEM environment variable with the value CHECKMOV.

The trace file contains a list of blocks in which the program has tried to execute a MOV and where there has been an overflow.

Example of the .LOG file generated:

```
Command  : wMov
Error src: length invalid, required=50, max allowed=44
Error dst: length invalid, required=50, max allowed=33
Action   : Action done
```

CHECKSTRINGS: Checks for overflows with the WLSTRCPY and WSETTEXT functions in the allocated memory

(Warning: this option slows programs down)

Runs programs with the NSDBGMEM environment variable with the value CHECKSTRING.

The trace file contains a list of blocks in which the program has tried to execute LSTRCPY and WSETTEXT in the allocated memory.

Example of the .LOG file generated:

```
Command  : wLStrCpy
Error src: OK
Error dst: length invalid, required=21, max allowed=12
Action   : Action done
```

CHECKALL: all the previous CHECK options

This option applies all the CHECKFILL, CHECKMOV and CHECKSTRING options

ADVERTUSER: Displays a message if there is a problem

Runs programs with the NSDBGMEM environment variable with the value ADVERTUSER

If a CHECK problem is detected, a message box is displayed with three options:

1. Truncate to the correct size.
2. Run the command as it stands.
3. Don't run the command.

Choose the desired option to continue the program.

ONERRORTRUNCLEN, ONERRORDOBADACTION, ONERRORABORT

These three options are used if the ADVERTUSER command has not been specified.

If a problem is detected, the selected mode of operation will be applied.

The SPYMEM memory monitor

This monitor lets you permanently display the amount of memory used by a program on the screen and monitor changes in real time.

Use

This mode is automatically enabled by setting the following environment variable:

```
set NSDBGMEM=SPYMEM
```

When the program is run, if the DLL NSxxSPYM.DLL is accessible, the SPY window appears.

Activate the memory monitor window, resize it if you so wish and place it in a corner of the screen. So that it is always visible, enable the "always on top" option and set the desired frequency for it to be refreshed. In order to prevent the title bar being displayed, enable the option "Hide title". To redisplay it, double-click on the window.

Run the application in the normal way.

Functionality

- A status bar indicating the amount of memory allocated per user and per application can be displayed.
- You can choose to display the user memory or the application memory.
- Pause function.
- Dialogue box displaying additional information and statistics on memory use.
- Automatic scale change.
- Name of the program being tested in the title bar.
- Refresh the window simply by double-clicking.

Key

The amount of memory allocated by the NEW command of the NCL application (the application itself or custom control) is displayed in yellow.

The amount of memory allocated by the NatStar or NSDK run-time is displayed in green.

MEMTRACE mode (with recompilation)

This mode provides you with the same information as DEBUGMEM mode but also enables you to find out on exactly which NCL line the problem has occurred.

This mode also provides you with a list of attempts to free memory blocks with an invalid pointer. It also allows you to find the location of the DISPOSE command in the NCL source file for each of these deallocations.

Use

Recompile projects with the /DNSMEMTRACE compilation option. Information appears in the NS-DESIGN information box.

Set the environment variable NSDBGMEM to the values seen previously.

The trace file

The names of the source files at fault and the line numbers appear in the trace file. You must then manually extrapolate the location in the NCL source code from the location in the C code generated (see appendix A).

Example of the file generated:

```
Program file mame  :
G:\prj\nsdk\GEN\INNT112W\DLL\TSTMEMTR.EXE
program start Time : 10:55:14

Command  : wFill
FileName : d:\PRJ\MEMORY\TSTMEMTR\WIN32\MAIN.c line:266
Error    : length invalid, required=50, max allowed=44
Pointer  : Allocated in file d:\PRJ\TSTMEMTR\WIN32\MAIN.c line:367
Action   : Action done

Command  : wMemDispose
FileName : d:\PRJ\MEMORY\TSTMEMTR\WIN32\MAIN.c line:321
Error    : invalid pointer :   FEFEFEh


Application memory leak
Unfreed blocks at application ending
------------------------------------
d:\PRJ\MEMORY\TSTMEMTR\WIN32\MAIN.c line=367 Size=    44 value=100609EA
d:\PRJ\MEMORY\TSTMEMTR\WIN32\MAIN.c line=358 Size=    33 value=10060AA2
d:\PRJ\MEMORY\TSTMEMTR\WIN32\MAIN.c line=340 Size=    11 value=1006093E
```

It can be seen from this example that:

- A pointer was allocated in line 367 of the file MAIN.C. A FILL attempt was made with an overflow in line 266 of the file MAIN.C.
- An invalid NCL DISPOSE instruction was called from the file MAIN.C in line

321. The pointer passed to this instruction had the hexadecimal value FE:FEFE and did not point to a valid memory area.

Three memory blocks were not freed on exiting the program and the location of their allocation is indicated.

⚠ The named or shared memory is not traced.

Look up from an NCL program of the various amounts of memory allocated

GETMEMORYINFOS% query API

This new API lets you find out the various amounts of memory allocated from an NS-DK (NatStar or NatWeb) program.

Use

Include the NCL NSDBGMEM resource file in an NCL project to be tested.

```
function GETMEMORYINFOS% (int Kind (2)) return int (4) external
'NS**MISC.GETMEMORYINFOS'
```

This function should be called with one of the following types of memory:

USER_MEMORY%	Returns the amount of memory allocated by the NEW command of NCL applications.
MAXUSER_MEMORY%	Returns the maximum value of the amount of user memory allocated.
APPLI_MEMORY%	Returns the amount of memory allocated per application (user program + NS-DK, NatStar or NatWeb Run-Time)
ALLOCATED_MEMORY%	Returns the amount of memory that the application is allocated in relation to the system.
MAXALLOC_MEMORY%	Returns the maximum value of the amount of memory allocated in relation to the system.
ALLOC_COUNT%	Returns the number of allocations (NEW) made in the application.
DESALLOC_COUNT%	Returns the number of deallocations (DISPOSE) made in the application.

The return value is expressed in bytes.

This function can be used to display the amount of memory allocated in the application or to automate memory loss checks.

API for verifying memory integrity

External CHECKMEMORY instruction 'NS**MISC.CHECKMEMORY'

Calling this instruction launches the memory integrity check.

SETMEMORYCHECKMODE int CheckMode (4) instruction

This instruction sets the memory integrity analysis mode to true or false.

SETMEMORYCHECKMODE instruction

When this instruction is set to TRUE%, all memory allocation or deallocation functions carry out an integrity analysis of the memory before being executed.

If a memory overwrite is detected, an error message is displayed and an NSDMEM.NSD trace file is written to the disk in the directory pointed to by the NS-TMP, TMP or TEMP environment variables.

When this instruction is set to FALSE%, it deactivates automatic analysis of the integrity of the memory.

Default value: FALSE%

APPENDICES

Appendix A, Steps to follow in order to find the NCL origin of a line of code generated in C

The trace files indicate the errors in the following form:

D:\nsdk\memory\tstmemtr\win32\main.c line=824 size=11

In this case the name of the resource at fault is: "MAIN".

Find the resource with the name MAIN in the NSDK (or NatStar) project There are two possible cases:

- The resource is of the window type (.SCR)
- The resource is of the library type (.NCL).

Case N° 1, the resource is of the SCR type

Example:

According to the trace file generated, two errors appear on the following lines.

```
d:\nsdk\memory\tstmemtr\win32\main.c line=824 size=11
d:\nsdk\memory\tstmemtr\win32\main.c line=836 size=33
```

```
switch (ID) {
        case 0:
            switch (msg) {
                case NS_MSG_INIT: {
Ligne 824 ->        iM1= (NS_LONG)wMemNew(0,11,"");
                    return 0;
                }
                case NS_MSG_TERMINATE: {
                    return 0;
                    }
            }
            break;
case ID_MAIN_PB_GETMEM3:
            switch (msg) {
                case NS_MSG_EXECUTED:
                    if (wDlgCheck1(wnd,ID_MAIN_PB_GETMEM3,100,120)) {
Ligne 836 ->            iM3= (NS_LONG)wMemNew(0,33,"");
                    }
                    return 0;
            }
            break;
```

Error in line 824

In this case, going back through the source code, you read 2 lines higher up, "case NS_MSG_INIT": This means that the code executed is in an INIT event.

Two lines further up, you read "case 0", (case depending on the « switch (ID)) the value 0 means that the event is executed in a window and not in a control.

The 1st error therefore occurred in the event INIT of the window MAIN.

Error in line 836

In this case, going back through the source code, you read 2 lines higher up, "case NS_MSG_EXECUTED". This means that the code executed is in an NCL EXECUTED event.

In going back 2 more lines, you read "case ID_MAIN_PB_GETMEM3", you need to break down the identifier "ID_MAIN_PB_GETMEM3".

- ID means Identifier.
- MAIN is the name of the resource (window).
- PB_GETMEM3 is the name of control.

The error therefore occurred in the EXECUTED event of the control PB_GETMEM3 of the window MAIN.

Case N° 2, the resource is of the NCL type

Example:

According to the trace file generated, two errors appear on the following lines.

```
d:\cvsprj\nsdk\memory\tstmemtr\win32\testmem.c  Line 20 Size=150
d:\cvsprj\nsdk\memory\tstmemtr\win32\testmem.c  Line 29 Size=80
```

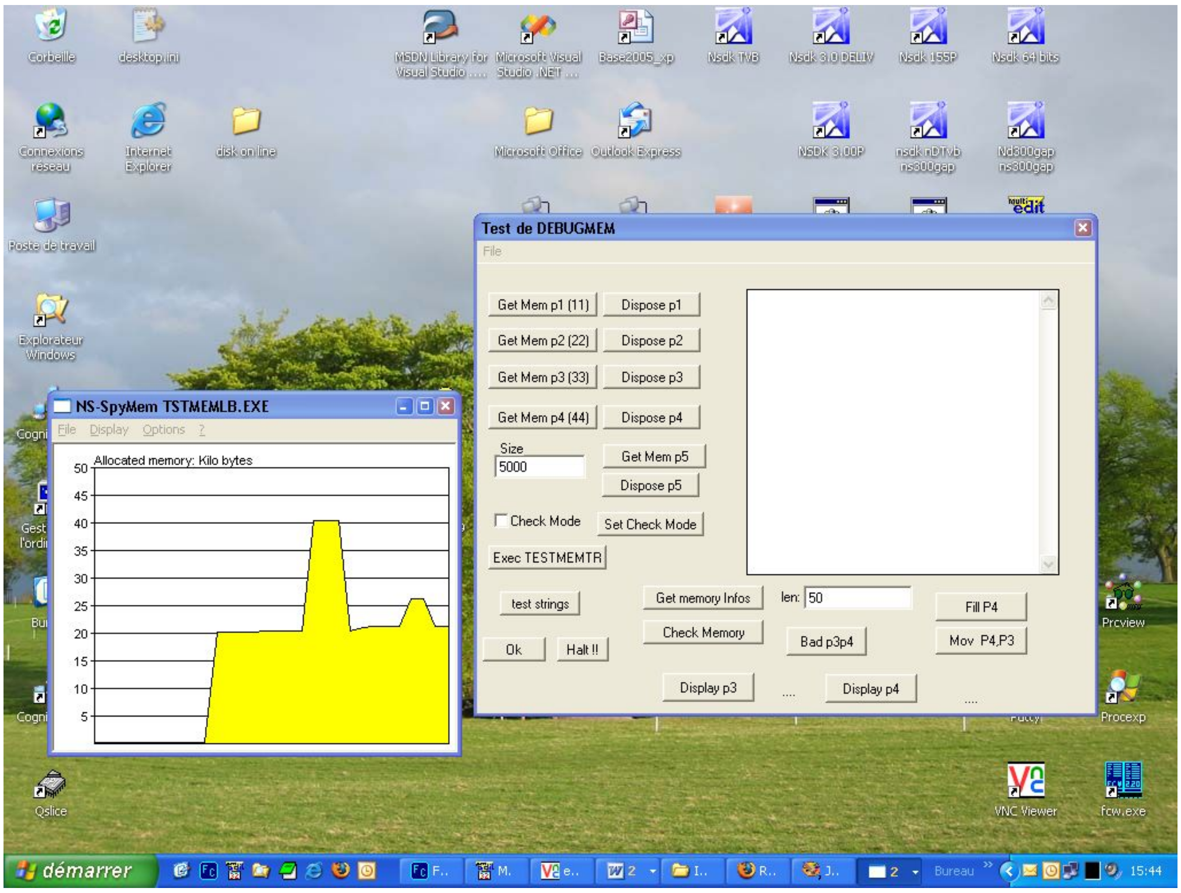
```
/*                                                               */
/* Library TESTMEM                                               */
/*                                                               */
NS_FN(void) riTESTMEM(void)
{
    iP= (NS_LONG)wMemNew(0,150,"");       <- Ligne 20
}
/*                                                               */
/* Instruction TESTMEMORY                                        */
/*                                                               */
NS_FN(void) TESTMEMORY(void)
{
    NS_LONG iP2;
    iP2= (NS_LONG)wMemNew(0,80,"");       <- Ligne 29
}
```

In the 1st case, the function has the same name as the library and has the "ri " prefix. This means that the error occurred during initialization of the library.

In the 2nd case, we can see directly that the error occurred in the instruction TESTMEMORY.

Display example of the SPYMEM window

Monitoring a program's memory consumption:


Test de DEBUGMEM
File
Get Mem p1 (11)
Dispose p1
Get Mem p2 (22)
Dispose p2
Get Mem p3 (33)
Dispose p3
Get Mem p4 (44)
Dispose p4
Size
5000
Get Mem p5
Dispose p5
Check Mode
Set Check Mode
Exec TESTMEMTR
test strings
Get memory Infos
len:
50
Fill P4
Check Memory
Bad p3p4
Mov P4,P3
Ok
Halt !!
Display p3
Display p4
NS-SpyMem TSTMEMLB.EXE
File Display Options ?
Allocated memory: Kilo bytes
50
45
40
35
30
25
20
15
10
5
démarrer
15:44