ADSP Crashman Usage Doc

August 27, 2015

Contents

[1.Crashman Scripts Location 2](#_Toc428451397)

[2. Requirements for Crashman: 2](#_Toc428451398)

[3. Crashman arguments explanation 2](#_Toc428451399)

[4. Crashman Usage from cmd prompt 3](#_Toc428451400)

[5. Crashman additional argument explanation with examples 3](#_Toc428451401)

[I. Crashman “-e” argument usage from cmd prompt 3](#_Toc428451402)

[II. Crashman “-lite” argument Usage from cmd prompt 4](#_Toc428451403)

[III. Crashman “-fullload” argument Usage from cmd prompt 4](#_Toc428451404)

[IV. Crashman “-sofile” argument Usage from cmd prompt 4](#_Toc428451405)

[V. Crashman “-use\_t32” argument Usage from cmd prompt 4](#_Toc428451406)

[VI. Crashman “-smmu64/-smmu32” arguments Usage from cmd prompt 5](#_Toc428451407)

[VII. Crashman “-load\_etm” arguments Usage from cmd prompt 5](#_Toc428451408)

[6. Crashman Output Folder Explanation 5](#_Toc428451409)

[7. How to verify if crashman command is successfully executed 6](#_Toc428451410)

[8. Auto Analysis Done by Crashman 6](#_Toc428451411)

1. Crashman Scripts Location

* <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman

## 2. Requirements for Crashman

* Access to crashman path or a local copy of it
* Python versions- **v2.7.8**(recommended),v2.5.2,v2.6.2,v2.7.2,v3.0.1,v2.7.6
* Perl versions- **v5.6.1** (recommended), v5.10.1, v5.8.7, v5.12.4, v5.14.2
* Hexagon Tools (optional, needed only for GDB based crashman)

## 3. Crashman arguments explanation

**python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> -e <CustomerprovidedOBJPath[Optional]>**

**Mandatory Arguments:** -t Targets Supported on Crashman   
 -d Dump File along with path   
 -o Output folder   
 -b ADSP Build Path Location

**Optional Arguments:**

|  |  |
| --- | --- |
| **Optional Argument** | **Explanation** |
| -e | Customer ELF Location |
| -lite | Crashman Lite Version (Only Dumps are loaded) |
| -gdb | GDB based crashman |
| -fullload | Loads the complete RAM Dumps |
| -sofile | Provide dynamic so files location. Presently Supports only 8952/8953/8937/8976/8996/8910 |
| -use\_t32 | Use T32 from a shared location |
| -smmu32/64 | Provide 32/64 bit vmlinux path |
| -load\_etm | ETM dump parsing |
| -vmlinux32/64 | Provide 32/64 bit vmlinux path |

***<TARGET>*** = Supported targets on crashman  
**8994/8974/9x25/9x35/8x10/8084/8x26/8996/8952/8992/8998/8976/8953/8937**

<***DumpFile>*** *=* Dump file along with path  
QPST dump [ex: DDRCS0.BIN, dram.lst\_0x00000000--0x2fffefff.lst] (Or)   
ADSP binary captured from trace (only ADSP memory region) [ex: LPASS\_Dumps.bin] (Or)   
ADSP binary extracted from EBI in case of windows (only ADSP memory region) [ex: ADSP0\_0x0C300000.bin] (Or) ADSP dump captured in elf format during SSR [ex: ramdump\_adsp\_1970-01-02\_00-05-06.elf]

***<Output\_Path>=*** Output folder where extracted crashman logs will be stored.

***<CRM\_Buildpath>*** = CRM Build path   
Ex: [\\cone\builds642\PROD\ADSP.BF.2.2.1-00001-M8610AAAAAAAZL-1](file:///\\cone\builds642\PROD\ADSP.BF.2.2.1-00001-M8610AAAAAAAZL-1)

***<CustomerprovidedOBJPath>*** =   
This argument is optional. Generally used for opendsp customer issues where elf files is given by customer.   
If this path argument is given in crashman command, the elf in the provided path will be loaded onto the trace. If not given, by default, the elf file in the ***CRM\_Buildpath*** is loaded.

## 4. Crashman Usage from cmd prompt

The Below crashman command, it will automatically detects start address and dump format and it start loading the dumps-

**a) pushd** <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
**b) python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath>**  
**Example**:  
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -t 8994 -d C:\DDRCS0.BIN -o C:\output -b [\\snowcone\builds698\PROD\ADSP.8994.2.6.1-00120-00374-1\adsp\_proc](file:///\\snowcone\builds698\PROD\ADSP.8994.2.6.1-00120-00374-1\adsp_proc)

## 5. Crashman additional argument explanation with examples

### Crashman “-e” argument usage from cmd prompt

If “–e” argument with elf\_path is given in crashman command, the elf in the provided path will be loaded onto the trace. If not given, by default, the elf file in the ***CRM\_Buildpath*** will be loaded.-

**a)** pushd<CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
**b)**python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> **-e <CustomerprovidedOBJPath>  
Example**:   
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -t 9x35 -d C:\DDRCS0.BIN -o C:\output -b [\\cone\builds634\PROD\ADSP.BF.2.4.1.c1-00021-M9635AAAAANAZL-1](file:///\\cone\builds634\PROD\ADSP.BF.2.4.1.c1-00021-M9635AAAAANAZL-1) -e [\\rover\hyd\_dspfw\ADSP\_Tools\TriageTeam\Users\Praveen\Testing\9x35\ELFS](file:///\\rover\hyd_dspfw\ADSP_Tools\TriageTeam\Users\Praveen\Testing\9x35\ELFS)

**“-e” argument Wrong usage:** elf name should not be given to argumentpython adspcrashman.py -t 8626 –d c:\dram\_0x0--0x5fffffff.bin –o c:\output -b [\\cone\builds632\PROD\ADSP.BF.2.2.c3-00010-M8626AAAAAAAZL-1](file:///\\cone\builds632\PROD\ADSP.BF.2.2.c3-00010-M8626AAAAAAAZL-1) **-e** **c:\customer\_elf\M8x26AAAAAAAAQ1234\_reloc.elf**

**“-e” argument Correct usage:**python adspcrashman.py –t 8626 –d c:\dram\_0x0--0x5fffffff.bin –o c:\output -b [\\cone\builds632\PROD\ADSP.BF.2.2.c3-00010-M8626AAAAAAAZL-1](file:///\\cone\builds632\PROD\ADSP.BF.2.2.c3-00010-M8626AAAAAAAZL-1) **-e** **c:\customer\_elf**

### Crashman “-lite” argument Usage from cmd prompt

Below crashman command, it will automatically detect start address and dump format and it loads only dumps and will not process the logs.

**a)** pushd<CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
**b)** python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> -e <CustomerprovidedOBJPath[Optional]> **-lite  
Example**:   
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -t 8994 -d C:\DDRCS0.BIN -o C:\output -b C:\lpass\_proc -lite

### Crashman “-fullload” argument Usage from cmd prompt

Below crashman command, it loads complete ram dumps ( load.cmm, DDRCS0.bin, DDRC1.bin etc. are necessary) to trace and process the logs

**a)** pushd<CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman   
**b)** python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> -e <CustomerprovidedOBJPath[Optional]> **-fullload <Complete\_RAM\_Dumps\_Path>**  
  
**Example**:  
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -t 8952 -o C:\Temp -d [\\lab5385\TriageTeam\Crashman\TestSuite\8952\mser\_171\DDRCS0.BIN](file:///\\lab5385\TriageTeam\Crashman\TestSuite\8952\mser_171\DDRCS0.BIN) -b [\\snowcone\builds677\PROD\ADSP.8952.2.6-00107-00000-1](file:///\\snowcone\builds677\PROD\ADSP.8952.2.6-00107-00000-1) -elf [\\lab5385\TriageTeam\Crashman\TestSuite\8952\mser\_171](file:///\\lab5385\TriageTeam\Crashman\TestSuite\8952\mser_171) -fullload [\\lab5385\TriageTeam\Crashman\TestSuite\8952\mser\_171](file:///\\lab5385\TriageTeam\Crashman\TestSuite\8952\mser_171)

### Crashman “-sofile” argument Usage from cmd prompt

Below crashman command, Once ADSP memory region is loaded to trace, it will try to detect the dynamic so files in the dumps from the \_rtld\_debug variable and loads the dynamic so files to the required memory region and if the so files are present in the given <SO\_File\_path> location and it further process the logs  
 **a)** pushd<CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman   
**b)** python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> -e <CustomerprovidedOBJPath[Optional]> **-sofile <SO\_Files\_Path>**

**Example**:  
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -t 8910 -d [\\lab5385\TriageTeam\Crashman\TestSuite\8910\evernote\_dead\_ramdump\_1\DDRCS0.BIN](file:///\\lab5385\TriageTeam\Crashman\TestSuite\8910\evernote_dead_ramdump_1\DDRCS0.BIN) -o c:\output -b [\\chronicle\zipbuild225\PROD\ADSP.BF.2.2.1-00023-M8610AAAAAAAZL-1](file:///\\chronicle\zipbuild225\PROD\ADSP.BF.2.2.1-00023-M8610AAAAAAAZL-1) -elf [\\lab5385\TriageTeam\Crashman\TestSuite\8910\jpege\_CAMERA\_JPEGE\_01\_01\_07](file:///\\lab5385\TriageTeam\Crashman\TestSuite\8910\jpege_CAMERA_JPEGE_01_01_07) -sofile [\\lab5385\TriageTeam\Crashman\TestSuite\8910\jpege\_CAMERA\_JPEGE\_01\_01\_07](file:///\\lab5385\TriageTeam\Crashman\TestSuite\8910\jpege_CAMERA_JPEGE_01_01_07)

### Crashman “-use\_t32” argument Usage from cmd prompt

This argument is to provide flexibility for users to have different T32 for different targets. So, user doesn’t need to install different T32 version to use crashman. Instead, they can give the shared T32 location path and it loads the dumps and process the logs using shared T32 tool -  
 **a)** pushd<CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman   
**b)** python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> -e[lf] <CustomerprovidedOBJPath[Optional]> **-use\_t32 <T32\_Shared\_Path>  
Example**:  
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -target 9x35 -dump C:\DDRCS0.BIN -output C:\output -build [\\cone\builds634\PROD\ADSP.BF.2.4.1.c1-00021-M9635AAAAANAZL-1](file:///\\cone\builds634\PROD\ADSP.BF.2.4.1.c1-00021-M9635AAAAANAZL-1) -e [\\rover\hyd\_dspfw\ADSP\_Tools\TriageTeam\Users\Praveen\Testing\9x35\ELFS](file:///\\rover\hyd_dspfw\ADSP_Tools\TriageTeam\Users\Praveen\Testing\9x35\ELFS) -use\_t32 [\\rover\hyd\_dspfw\ADSP\_Tools\T32](file:///\\rover\hyd_dspfw\ADSP_Tools\T32)

### Crashman “-smmu64/-smmu32” arguments Usage from cmd prompt

With this new argument complete ram dump will be loaded and SMMU page table translation is also taken care. This will be used for debugging content in SMMU shared buffer/ dynamic heap .we need to have complete ram dumps ( load.cmm, DDRCS0.BIN, DDRC1.BIN etc. are necessary) and vmlinux file used to collect ram dump-  
 **a)** pushd<CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman   
**b)** python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> -e <CustomerprovidedOBJPath[Optional]> **-smmu64/smmu32 <64/32Bit vmlinux path>**

**Example**:  
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -t 8952 -o C:\Temp -d [\\lab3079\Sahara\matting\_171\DDRCS0.BIN](file:///\\lab3079\Sahara\matting_171\DDRCS0.BIN) -b [\\snowcone\builds677\PROD\ADSP.8952.2.6-00107-00000-1](file:///\\snowcone\builds677\PROD\ADSP.8952.2.6-00107-00000-1) -smmu64 [\\holi\builds539\TEST\LA.BR.1.3.1-27101-8952.1-1\LINUX\android\out\target\product\msm8952\_64\obj\KERNEL\_OBJ\vmlinux](file:///\\holi\builds539\TEST\LA.BR.1.3.1-27101-8952.1-1\LINUX\android\out\target\product\msm8952_64\obj\KERNEL_OBJ\vmlinux)

### Crashman “-load\_etm” arguments Usage from cmd prompt

ETM dump analysis will be done by giving this argument  
if we have etr/etm.bin file in the dump path –load\_etm argument will load the etr/etm dump to trace memory  
if we have complete dump path and –vmlinux64/32 will extract etm memory from the dumps and loads to trace memory -  
 **a)** pushd<CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman   
**b)** python adspcrashman.py -t <TARGET> -d <DumpFile> -o <Output\_Path> -b <CRM\_Buildpath> -e <CustomerprovidedOBJPath[Optional]> **-load\_etm –vmlinux64/32 <64/32bit vmlinux path>**

**Example**:  
pushd <CRM\_BUILD\_PATH>\adsp\_proc\qdsp6\scripts\crashman  
python adspcrashman.py -t 8952 -o C:\Temp -d [\\lab5385\TriageTeam\Crashman\TestSuite\8952\Port\_COM82\_log2ETM\DDRCS0.BIN](file:///\\lab5385\TriageTeam\Crashman\TestSuite\8952\Port_COM82_log2ETM\DDRCS0.BIN) -b [\\snowcone\builds732\INTEGRATION\ADSP.8952.2.6-00137-00000-1](file:///\\snowcone\builds732\INTEGRATION\ADSP.8952.2.6-00137-00000-1) -load\_etm -vmlinux64 [\\diwali\builds534\TEST\LA.BR.1.3.1-34103-8952.1-5\LINUX\android\out\target\product\msm8952\_64\obj\KERNEL\_OBJ\vmlinux](file:///\\diwali\builds534\TEST\LA.BR.1.3.1-34103-8952.1-5\LINUX\android\out\target\product\msm8952_64\obj\KERNEL_OBJ\vmlinux)

## 6. Crashman Output Folder Explanation

**Extracted crashman logs contains-  
adspcrashman\_click.bat:** If weDouble-click batch file it will load only the dumps and will not process logs  
**Logs:** All the Extracted logs will be present in this folder.

* **DSPAnalysis.txt:** This file will contain auto-analysis of whole dump and will give all the general related info in one text file. This contain information about crash thread, cause of error, instructions at crash location, registers set, heap information, F3 error/failure messages and last few ulogs,f3 logs etc.
* **f3log.txt:** Extracted f3 logs
* **apr\_logs.txt:** Extracted apr logs
* **Crashman\_Log.txt:** Crashman Processing logs
* **Adsp\_Dump.bin:** ADSP memory dump
* **adsppm:** Extracted adsppm logs
* **Def\_Heap:** Default memory heap logs
* **Def\_Heap\_forSensors\_img:** Default sensor memory heap logs
* **Qurt\_logs:** Extracted QURT related logs in this folder.
* **SensorF3Logs:** Extracted sensor f3 logs.
* **Ulog:** Extracted Ulog related logs in this folder.
* **temp:** All temporary files will be stored in this folder.

7. How to verify if crashman command is successfully executed  
 - DSP Analysis.txt will be created in output folder.   
 - In T32 window, on right we see AREA.view DSPAnaly will be created once processing is done.

## 8. Auto Analysis Done by Crashman

* Validates timestamp even before loading the dumps. The Engg timestamp in the provided dumps is compared with the CRM ELF or Customer provided ELF file.
* Creates a copy of ADSP image dump region (from the entire RAM dump) in the output log directory.
* Entire RAM dumps is loaded for targets in order to debug SMMU region.
* Only ADSP region dump is loaded for all targets in order to make the process fast.
* Crashman so file argument will try to detect the dynamic so files in the dumps from the \_rtld\_debug variable and loads the dynamic so files to the required memory region
* Crashman lite will automatically detect start address and dump format and it loads only dumps and will not process the logs.
* Generates text files with following views: Name of crashed thread, Cause for crash, Call stack of crashed thread, register set of crashed thread, heap check, stack check, core dump, chicken bits value, disassembled code of crashed thread, apr logs etc.
* Print error/underrun/overflow/unsupported F3 messages and also print last few F3 messages.
* Merges available ULogs and print last few ULog messages.
* Generates DSPAnalysis.txt file that has below info.

1. It provides Crashman command
2. Whether the dumps are valid or not based on time stamp/cache flushed.
3. Core dump info is updated
4. Updates Running call stack and Stack walker call stack
5. Crash SW Thread name. Though the crash shows in err\_x thread in dumps, actual thread in which crash has occurred is traced and shown.
6. Call stack of the crashed thread along with its local variables.
7. If the crash happened because of error fatal in any SW thread other than err\_x, displays the error message, file name at which crash occurred and the params provided to the error message.
8. Informs whether stack is corrupted or not.
9. Informs whether heap is corrupted or not both for GuestOS and SensorsPD.
10. Informs Memory leaks/Exhaustion if any from the dumps.
11. Disassembled code where the crash has happened.
12. Register set of the crashed thread.
13. Merged ulog messages and Last few F3 logs for GuestOS and Sensor PDs
14. Last Few error messages of F3logs